

NAVAL POSTGRADUATE SCHOOL

Monterey, California



THESIS

AMPHIBIOUS OPERATIONS IN A VIRTRUAL ENVIRONMENT

by

Didier A. Le Goff

March 1997

Thesis Advisor:
Thesis Co-Advisor:

Michael J. Zyda
John S. Falby

Approved for public release; distribution is unlimited.

19971125 026

THIS QUALITY INSPECTED 4

REPORT DOCUMENTATION PAGE			Form Approved OMB No. 0704-0188	
Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time reviewing instructions, searching existing data sources gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden to Washington Headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188), Washington, DC 20503.				
1. AGENCY USE ONLY (Leave Blank)		2. REPORT DATE March 1997		3. REPORT TYPE AND DATES COVERED Master's Thesis
4. TITLE AND SUBTITLE Amphibious Operations in a Virtual Environment.			5. FUNDING NUMBERS	
6. AUTHOR(S) Le Goff, Didier, A.				
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Naval Postgraduate School Monterey, CA 93943-5000			8. PERFORMING ORGANIZATION REPORT NUMBER	
9. SPONSORING/ MONITORING AGENCY NAME(S) AND ADDRESS(ES)			10. SPONSORING/ MONITORING AGENCY REPORT NUMBER	
11. SUPPLEMENTARY NOTES The views expressed in this thesis are those of the authors and do not reflect the official policy or position of the Department of Defense or the United States Government.				
12a. DISTRIBUTION / AVAILABILITY STATEMENT Approved for public release; distribution is unlimited.			12b. DISTRIBUTION CODE	
13. ABSTRACT (Maximum 200 words) More than 80 percent of recent, real world, naval operations have taken place in the littoral; over half have employed amphibious units. However, up till now, no simulation developed at the Naval Postgraduate School had the capability to exercise any type of naval amphibious operation. Previous simulations lacked the necessary amphibious ship and landing craft models. Second, a method for nesting mounted entities did not exist. The approach taken was to develop a general algorithm for dynamically mounting, unmounting and nesting entities. Secondly, amphibious ship and landing craft models were developed incorporating a simple hydrodynamic models for use with the LPD-17 and Landing Craft Air Cushion (LCAC) vehicles. Finally, real-time collision detection was implemented to ensure realistic interaction between all entities. The result is a stand-alone, 3-D, virtual environment (VE) which simulates landing craft embarkation operations between a mother ship (LPD-17 class) and an LCAC, and allows embarked entities to walk through the 50,000 polygon LPD model in real-time (7-15 frames per second). The simulation is further enhanced by realistic wave response, based on the Pierson-Moskowitz spectrum, by all ocean borne vehicles. Lastly, the use of the high level EasyScene 3.0 API allowed the application to be written in approximately 30 percent few lines of code than otherwise possible.				
14. SUBJECT TERMS NPSNET, Easy Scene, real-time, 3D, visual simulation, Performer, interactive, virtual world, PVS, Potentially Visible Sets, mounting humans entities, ocean model, tool kit.			15. NUMBER OF PAGES 88	
			16. PRICE CODE	
17. SECURITY CLASSIFICATION OF REPORT Unclassified	18. SECURITY CLASSIFICATION OF THIS PAGE Unclassified	19. SECURITY CLASSIFICATION OF ABSTRACT Unclassified	20. LIMITATION OF ABSTRACT UL	

Approved for public release; distribution is unlimited.

**AMPHIBIOUS OPERATIONS
IN A VIRTUAL ENVIRONMENT**

Didier A. Le Goff
Lieutenant, United States Navy
B.A., Northwestern University, 1988

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN COMPUTER SCIENCE

from the

NAVAL POSTGRADUATE SCHOOL

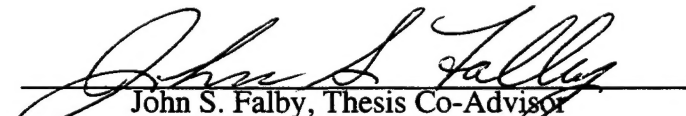
March 1997

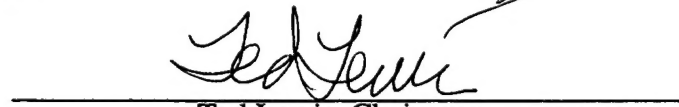
Author:


Didier A. Le Goff

Approved by:


Michael J. Zyda, Thesis Advisor


John S. Falby, Thesis Co-Advisor


Ted Lewis, Chairman,
Department of Computer Science

ABSTRACT

More than 80 percent of recent, real world, naval operations have taken place in the littoral; over half have employed amphibious units. However, up till now, no simulation developed at the Naval Postgraduate School had the capability to exercise any type of naval amphibious operation. Previous simulations lacked the necessary amphibious ship and landing craft models. Second, a method for nesting mounted entities did not exist.

The approach taken was to develop a general algorithm for dynamically mounting, unmounting and nesting entities. Secondly, amphibious ship and landing craft models were developed incorporating a simple hydrodynamic model for use with the LPD-17 and Landing Craft Air Cushion (LCAC) vehicles. Finally, real-time collision detection was implemented to ensure realistic interaction between all entities.

The result is a stand-alone, 3-D, virtual environment (VE) which simulates landing craft embarkation operations between a mother ship (LPD-17 class) and an LCAC, and allows embarked entities to walk through the 50,000 polygon LPD model in real-time (7-15 frames per second). The simulation is further enhanced by realistic wave response, based on the Pierson-Moskowitz spectrum, by all ocean borne vehicles. Lastly, the use of the high level EasyScene 3.0 API allowed the application to be written in approximately 30 percent few lines of code than otherwise possible.

TABLE OF CONTENTS

I.	INTRODUCTION	1
A.	BACKGROUND	1
1.	NPSNET	1
2.	Shiphandling Training Simulator.....	2
3.	Damage Control Virtual Environment Trainer.....	2
4.	Mounted Networked Human Entities	2
B.	MOTIVATION.....	3
1.	Diminished Training Opportunities	3
2.	Applying Virtual Reality.....	4
C.	OBJECTIVE	5
D.	APPROACH	6
E.	SUMMARY OF CHAPTERS	7
II.	PREVIOUS RESEARCH	9
A.	SHIPHANDLING TRAINING SIMULATOR.....	9
B.	DAMAGE CONTROL VIRTUAL ENVIRONMENT TRAINER.....	10
C.	MOUNTED NETWORKED HUMAN ENTITIES.....	16
III.	SOFTWARE ARCHITECTURE	19
A.	BACKGROUND	19
1.	High Speed Rendering	19
2.	Additional Functionality	20
3.	NEXTGEN and NPSNET V	20
B.	IRIS PERFORMER API.....	21
1.	Multiprocessing.....	21
2.	Hierarchical Database	23
C.	EASYSCENE 3.0 API.....	25
1.	Structure.....	25
2.	Run-time Interface	28

D.	SUMMARY	30
IV.	EASYSCENE 3.0 MARITIME MODULE	33
A.	INTRODUCTION	33
B.	OCEAN MODEL.....	33
C.	OCEAN SPECTRA	35
D.	esSea CONTRUCTION.....	36
E.	SUMMARY	37
V.	COLLISION DETECTION	39
A.	INTERSECTION TESTING	39
1.	Basic Mechanics	39
2.	Run Time	41
B.	INTERSECTION AND COLLISION MASKS	42
C.	IMPLEMENTATION.....	43
1.	LPD	44
2.	LCAC.....	44
3.	Human.....	45
D.	CONCLUSION.....	47
VI.	MOUNTING ENTITIES	49
A.	INTRODUCTION	49
B.	DIS RESTRICTIONS.....	50
C.	MOUNTING ALGORITHM.....	51
D.	CONCLUSION.....	54
VII.	MODELS	57
A.	INTRODUCTION	57
1.	LPD-17.....	57
2.	Scenes from the Amphibious Simulator	59
VIII.	CONCLUSION	63
A.	RESULTS	63
B.	FUTURE WORK.....	64

APPENDIX	67
A. USER CONTROLS	67
B. RUNNING THE SIMULATOR.....	68
LIST OF REFERENCE	69
BIBLIOGRAPHY	71
INITIAL DISTRIBUTION LIST	73

LIST OF FIGURES

1.	The Antares pier side.	10
2.	PVS example.	12
3.	Sailor in CIC.	14
4.	Firefighter in the engine room.	15
5.	View from the Antares' bridge wing.	16
6.	Performer run-time loop.	22
7.	Performer process pipeline.	23
8.	Simplified scene graph for the amphibious simulator.	24
9.	esObject manipulations.	27
10.	EasyScene interface with the Performer run-time loop. After [CSIB96].	29
11.	High level class hierarchy for the amphibious simulator.	32
12.	Maritime Module Ocean Model.	34
13.	esSea and constituent waves. After [CSIB96].	37
14.	Creation of an ocean with W waves and F frequencies.	38
15.	Collider example.	40
16.	Cube colliding with a wall object.	40
17.	Cube and wall are not colliding.	41
18.	Collisions based on intersection and collision masks.	43
19.	LCAC collision zones.	45
20.	Human entity dimensions.	46
21.	Mounting entities through local coordinate systems.	51
22.	LCAC mounted on LPD.	52
23.	Deeply nested human mounted on LCAC.	53
24.	Complete LPD model hierarchy viewed in MultiGen.	58
25.	Model hierarchy for amphibious simulator.	58
26.	Side view of LPD-17.	59
27.	Human's view of LCAC in well-deck.	60

28.	Human on LPD's flight deck.	60
29.	Chart table in CIC.	61
30.	Command consoles in CIC.	62

LIST OF TABLES

1.	Potential Visibility List	12
2.	EasyScene to Performer coordinate conversions.	25
3.	esModule callbacks	28
4.	Amphibious simulator's module structure	31
5.	Marine Module Sea State Defaults	34
6.	Dynamic sea / performance trade off.	35
7.	Amphibious simulator controls.	67

I. INTRODUCTION

A. BACKGROUND

1. NPSNET

The NPSNET Research Group at the Computer Science Department of the Naval Postgraduate School in Monterey, California, is a group of faculty, staff and students working in various areas of networked virtual environments. The group's main software system is the Naval Postgraduate School Networked Vehicle Simulator (NPSNET), which integrates the research components into a networked computer simulation of up to 300 players using currently available off-the-shelf (Silicon Graphics, Inc.) workstations and networking technology [NPSN96]. NPSNET is capable of simulating articulated humans, stealth objects, surface and subsurface sea vessels, and air and ground vehicles in the DIS networked virtual environment.

Most of the progress made in NPSNET has been in the areas of air and ground combat simulations. However, foundations have been laid for naval simulation in NPSNET. This research continues the work of several previous theses which are part of the NPSNET Ship Program. The Ship Program includes the Shiphandling Training Simulator (SHIPSIM), Damage Control Virtual Environment Trainer (DC VET) and Networked Mounted Human Entities. This literature is briefly mentioned here but is covered in greater detail in the next chapter.

The goal of this thesis is to adapt the conceptual advances previously made in the area of shipboard simulations and incorporate them into a stand alone application capable of simulating amphibious operations. The interaction of a Landing Craft Air Cushion and an

amphibious mother ship, LPD-17 class, is demonstrated in the process of examining this work. Many other types of amphibious operations can be simulated in this fashion. Some of these are briefly discussed but their implementation is left as future work.

2. Shiphandling Training Simulator

Shiphandling Training Simulator or SHIPSIM addresses the lack of portable, real-time, interactive simulations for use in the training of junior naval officers in hazardous at sea evolutions such as underway replenishments, mooring to a buoy and multi-ship seamanship exercises[NOBL95]. Currently used trainers are few in number, fixed and extremely expensive to run. As a result this equipment is used mostly to hone the already developed skills of prospective Department Heads, and Executive and Commanding Officers. SHIPSIM uses commercially available Silicon Graphics, Inc. products and DIS network protocol to develop a networked shiphandling trainer suitable for shipboard junior officer use.

3. Damage Control Virtual Environment Trainer

Damage Control Virtual Environment Trainer or DC VET is a stand-alone, interactive, real-time, networked, virtual environment which allows a human entity to move about a ship. Its purpose is to train personnel in shipboard forfeiting techniques as well as to serve as a method for indoctrinating new personnel to the general layout and functions of a naval vessel. The user can manipulate certain objects such as fire nozzles, valves, doors, etc. Shipboard casualties such as steam leaks and fuel oil fires can be started to train individuals or teams in damage control procedures. [KING95] and [OBYR95]

4. Mounted Networked Human Entities

This work combined the two previous stand-alone products into one simulation and fully incorporated it into NPSNET. The result was a distributed environment in which a human entity

can enter into, and interact with, buildings or ships, which themselves are virtual environments, to conduct training.[STEW96]

B. MOTIVATION

1. Diminished Training Opportunities

The old philosophy of “train as you fight and fight as you train” has served the military well for a long time. It means that you practice by doing. The more realistic the training the more it pays off. In terms of amphibious operations this includes mobilizing troops and getting ships underway for extended periods of time, conducting live gunnery exercises, assaulting actual beaches and performing dangerous ship-to-shore movements. In addition to all this, a credible opposition force must be provided in order to complete the picture. This training cycle must be repeated with specific frequency to achieve and maintain proficiency.

Several factors have in recent years conspired to make it increasingly difficult to meet these training requirements. The most obvious is financial. As the defense budget gets smaller, available underway time is reduced. Maintenance costs are taking a larger percentage of total unit operating budgets leaving less for modernization and training.

Another factor is environmental. Live training traditionally conducted in many Navy/ Marine Corps facilities has been stopped or severely curtailed. For example, in large parts of California and Hawaii live fire and amphibious landing areas have been placed off limits during much of the year either for fire prevention or protection of wetlands and nesting grounds. Also, throughout the Navy, strict EPA regulations on hazardous material handling, refueling and emissions control must be enforced. These environmental factors

are by no means being criticized as excessive or unnecessary. Compliance with them, however, is another reason why live training is getting more expensive and difficult to accomplish.

The last factor is political. In the past, training was also conducted while overseas on deployments to maintain proficiency. These exercises were conducted on U.S controlled forward bases throughout the world. The number and suitability of these bases has been shrinking since the end of the Cold War. By far the largest such bases in the Pacific Fleet were Subic Bay, Republic of Philippines and Okinawa, Japan. One is already gone and we are in jeopardy of losing the other. To offset this loss exercises are held unilaterally or with combined forces in an area controlled by a host country. However, these are not permanent arrangements and availability is subject to short notice change.

As a result of these factors commanding officers are forced to turn to other training methods to keep personnel qualified. This work proposes using virtual environments as one such method.

2. Applying Virtual Reality

“VR applications become practical if creating a simulated environment is cheaper than placing the user in an actual physical environment, or if the physical environment is otherwise inaccessible.”[LOCK95] The cost of the actual physical environment is constantly increasing. Its accessibility is determined by many of the factors discussed above as well as other strategic factors. In most cases the area in which actual operation will take place will not be available for use in training. Virtual reality allows the simulation of significant environmental aspects such as terrain, ocean conditions, weather and man-made structures all tailored to better prepare our personnel. A computer can populate the simulation with entities, human or otherwise, to increase realism as well as provide an enemy force against which to train. Virtual reality is scalable from a single user on a stand-alone workstation to a networked version allowing multiple users to share the same

environment. Commanders can participate in high level decision making dealing only with coordination issues or can evaluate the actions of subordinate units at a level of detail not possible in real life.

C. OBJECTIVE

This research project's main purpose is to demonstrate the feasibility of conducting amphibious training in a virtual environment. Particular interest is placed in developing the ability to embark and disembark amphibious landing vehicles from the welldeck of a mother/command ship. These vehicles are used to conduct ship-to-shore movements carrying troops who may also be a part of the simulation. With this accomplished follow on work can proceed at several levels.

Onboard ship, individual watchstanders can train in emergency as well as normal procedures without getting the ship underway and in complete safety. For example, welldeck officers need to train/rehearse the proper sequence of events involved in launching and recovering vehicles. Each mission is slightly different. Vehicles need to be staged in proper order, the welldeck must be ballasted to a certain depth depending on types of vehicles being deployed, proper techniques for controlling vehicles in and out of the well need to be learned, etc. In the Combat Information Center (CIC), the Primary Control Officer (PCO) needs to coordinate assault waves as they leave the ship and control the landing craft through the boat lanes onto the beach to arrive according to a pre-determined timeline. Later on-call waves and backloads also need coordination.

At the staff level, commanders can practice coordinating large scale assaults using any of his normally available assets. Alternate load plans and beach landing sites may be tested, again without getting an entire Task Force underway.

Simulations such as these will never, of course, completely remove the need for live training. However, they will reduce the need to get underway and will make these at sea periods safer because crews will be better prepared.

D. APPROACH

This work, although related to the previous research in content, is not part of NPSNET. Instead, as mentioned earlier, it is a stand alone simulation with EasyScene 3.0 (es3.0) used for development. Es3.0 is a real time 3-D simulation development tool kit by Coryphaeus Software, Inc., and is built on top of SGI's Performer Library. Both these application programmer's interfaces (API) will be discussed in greater detail in a later chapter.

Why a stand alone simulation? Starting "from scratch" affords three advantages over the current NPSNET implementation.

First, the es3.0 tool kit allows the simulation to be built in a completely modular fashion. Modules for new entities or new functionality for existing entities can be created and just "plugged in" to the existing application. These modules, in many cases, can be written with little or no knowledge of the inner workings of the rest of the simulation. Project management and future expansion are therefore much easier.

The second advantage is related to the first. Although this work will not be compatible with NPSNET IV, its organization is very similar to NPSNET V, NPSNET's proposed successor. NPSNET V will be constructed using a custom developed tool kit, NextGen, similar in many ways

to es3.0. In other words, this work can be readily ported to NextGen, once completed, and incorporated as a plug-in module.

Lastly, the high level nature of es3.0 allows the application to be written in approximately 30% fewer lines of code than would otherwise be possible without losing the ability to manipulate low level details when required.

E. SUMMARY OF CHAPTERS

- Chapter II includes a more detailed discussion of previous research.
- Chapter III is an overview of the proposed NPSNET V, NextGen.
- Chapter IV provides a look at the Performer API and EasyScene 3.0 Tool Kit.
- Chapter V discusses the EasyScene 3.0 Maritime Module.
- Chapter VI examines the use of collision detection and techniques for mounting entities inside one another.
- Chapter VII an overview of the LPD-17 and LCAC models used in this work, as well as screen shots of the simulation.
- Chapter VIII contains a critique of this implementation's shortfall's and a discussion of future work needed in this effort.
- The appendix contains a user's manual for the simulation.

II. PREVIOUS RESEARCH

This chapter gives an overview of work previously done in the NPSNET Ship Project which is of direct value to this research. The pertinent concepts are summarized. Many will be covered in further detail in later chapters.

A. SHIPHANDLING TRAINING SIMULATOR

The Shiphandling Training Simulator (SHIPSIM) is a stand-alone, interactive, networked, real-time virtual environment for maneuvering a ship in various shiphandling evolutions, such as restricted water piloting, mooring to a buoy, and underway replenishments (UNREP). It was designed to train junior officers to conn U.S naval vessels and reduce at sea collisions [NOBL95]. Figure 1 shows the ship at the pier.

SHIPSIM gives the user the capability to maneuver through the virtual environment using a Graphical User Interface (GUI) which interprets ship control functions of rudder angles and engine orders. The physics used to model the movement of the ships through the water is based on a twin-screwed, single rudder ship. "Twisting" (off-axis thrust used to tighten a turning radius) is taken into account. The model, however, only deals with two dimensions and three degrees of freedom (surge, sway and heading). Sink, pitch and roll are not incorporated.

The use of Distributed Interactive Simulations (DIS) network protocol allows many users to participate in the same simulation and engage in multi-ship tactical and seamanship exercises. The different users may be connected via a standard ethernet LAN or across the

Internet [LOCK94]. The maximum number of participants in the simulation is limited by the capabilities of the user's workstation and network loading

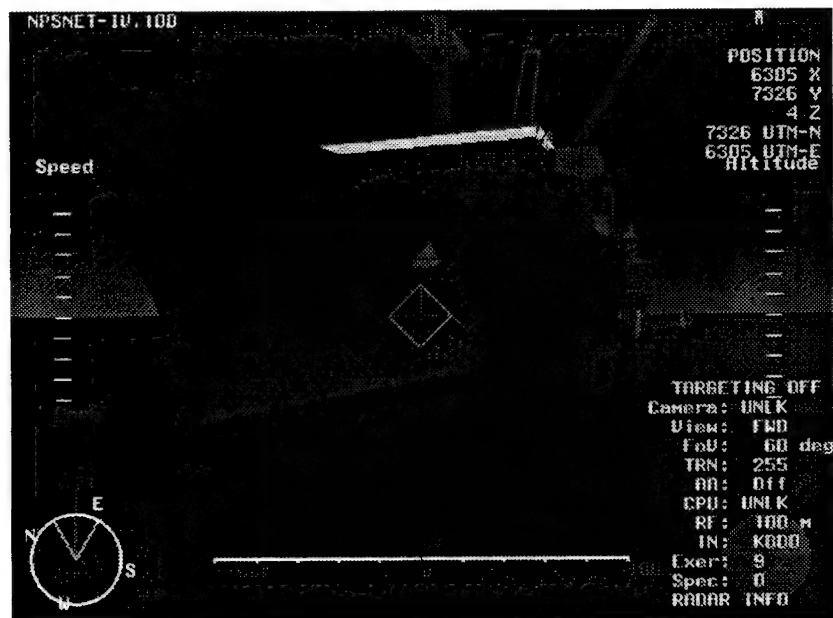


Figure 1: The Antares pier side.

B. DAMAGE CONTROL VIRTUAL ENVIRONMENT TRAINER

The Damage Control Virtual Environment Trainer (DC VET) was designed to facilitate the training of individuals and teams in shipboard firefighting procedures and general shipboard familiarization through an interactive, networked simulation [KING95]. It was subsequently modified with the addition of an articulated human called *Jack*¹, used to represent players in the virtual environment [OBYR95] as well as sounds in order to enhance user immersion.

1. The *Jack* Motion Library was developed at the University of Pennsylvania.

[KING95] and [OBYR95] present several key concepts which are important and applicable to this research. The DC VET uses the SHIPSIM model of a proposed roll-on/roll-off commercial ship, the Antares, which was built for Naval Sea Systems (NAVSEA) by Advanced Marine Vehicles. The base model was created using MultiGen modeling software. The interior was redesigned to more closely resemble a naval vessel and spaces such as combat information center (CIC), engine room and pilot house were added. DC VET loads the model and allows the user to move about the ship. In addition, the user, in the form of the *Jack* entity, can conduct firefighting training in the engine room and open and close doors and valves throughout the ship.

During the design, the concepts of Potentially Visible Sets (PVS) [AIREY90] and Level of Detail (LOD) were utilized to reduce the size of the model before reaching the graphics pipeline and real-time rendering was achieved. Normally, the culling process of an application goes through the entire scene database and reduces its size through the use of clipping planes and hidden surface removal algorithms before being sent to the rendering process. Sometimes, given a large enough database, even this is not enough to ensure acceptable, at least 6 frames per second, performance. In the case of the Antares, many areas which would still be rendered after culling are not “potentially visible” to the user at any given time due to opaque objects such as floors, doors, walls and bookcases. PVS attempts to remove these areas from culling and rendering consideration and therefore increase performance. PVS divides the database into various volumes called cells. To each cell is attached a list of other cells which are potentially visible to a user in that original cell. Now only cells previously determined to be visible from the user’s cell are sent to the

culling and rendering processes. As the user moves, new visible cells are added to the list and older non-visible cells are removed. This concept was initially envisioned for walkthroughs of architectural databases, especially of buildings. Each room is a cell and other rooms are visible only through windows or doors. From Figure 2, you can see that an observer in Office A can only see into Office D. No other rooms are “potentially visible” according to Table 1. Only Offices A and B, therefore, need to be considered for culling and rendering.

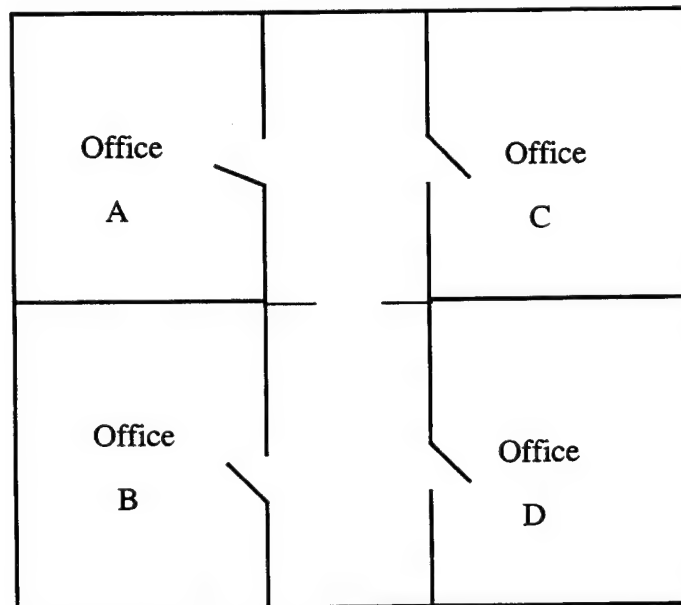


Figure 2: PVS example.

Table 1: Potential Visibility List

From	Visible
Office A	C
Office B	D

Table 1: Potential Visibility List

From	Visible
Office C	A
Office D	B

To implement the PVS algorithm in DC VET, the ship is partitioned into useful size compartments such as engine room, bridge, CIC etc. and each cell is numbered. The cells are then provided, in the form of an array, a list of all the potentially visible cells from itself. As the user travels from one part of the ship to another, the application dynamically swaps out those cells considered no longer visible and swaps in the new potentially visible set. Although this dynamic restructuring of the scene graph is expensive, the result, in this application, is an up to ten fold increase in rendering speed; from 2 to 20 frames per second.

The ability to pass through walls and other obstructions greatly reduces the credibility a simulation. For this reason collision detection is an important part of this project. Two types of collision detections are implemented in DC VET: deck or ground detection which allows the user to walk around the ship and up and down stairs, and object collision to simulate interactions with solid obstructions such as walls. A user's interaction with the virtual environment is accomplished through another form of intersection testing called picking. By using picking in DC VET, doors can be opened, valves cycled and objects moved. Collision detection, its different uses, and their implementations will be examined more thoroughly in a later chapter. Figures 3 - 5 show various scenes from DC VET.

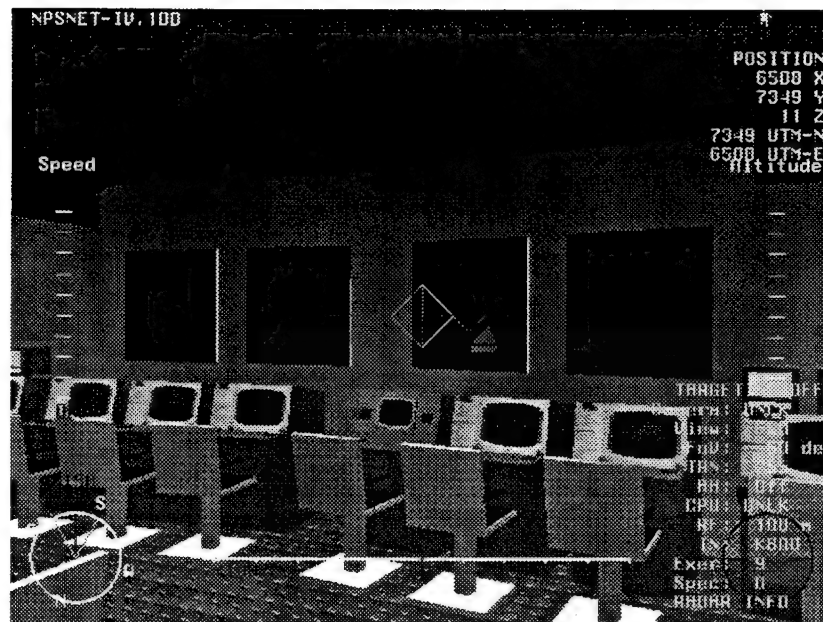


Figure 3: Sailor in CIC.

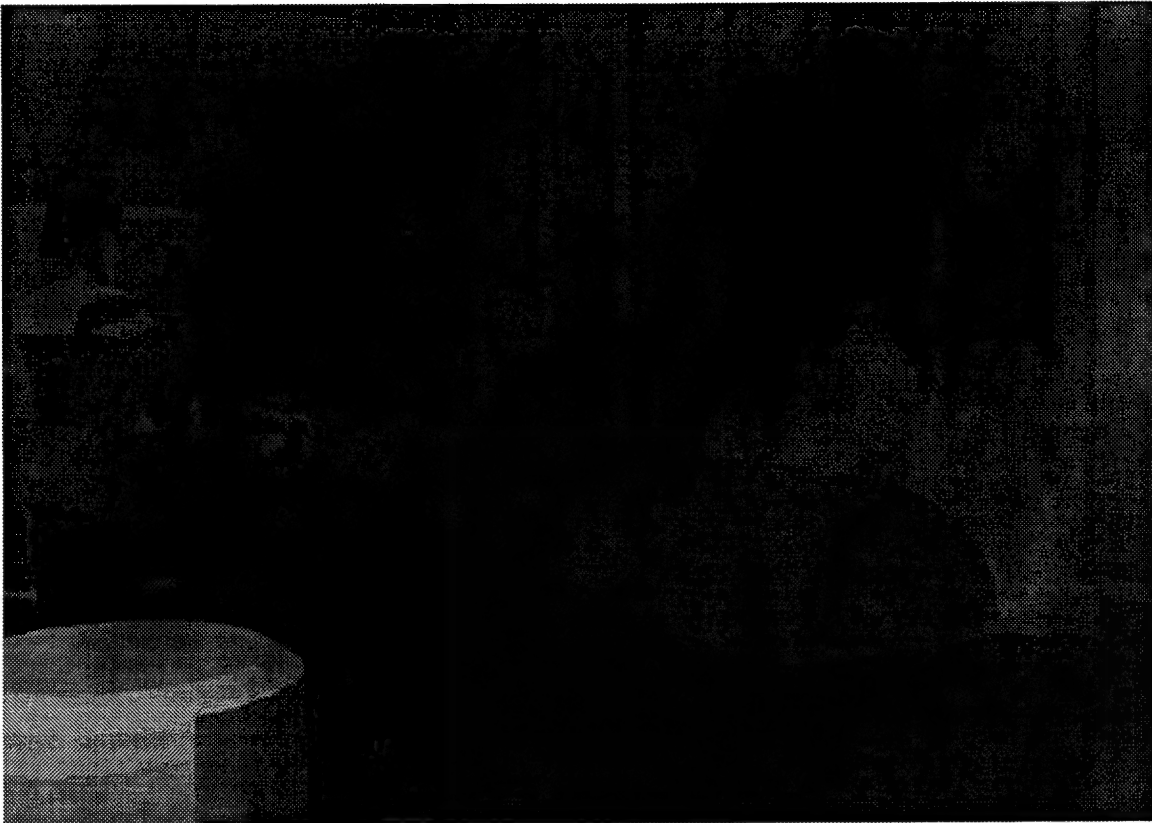


Figure 4: Firefighter in the engine room.

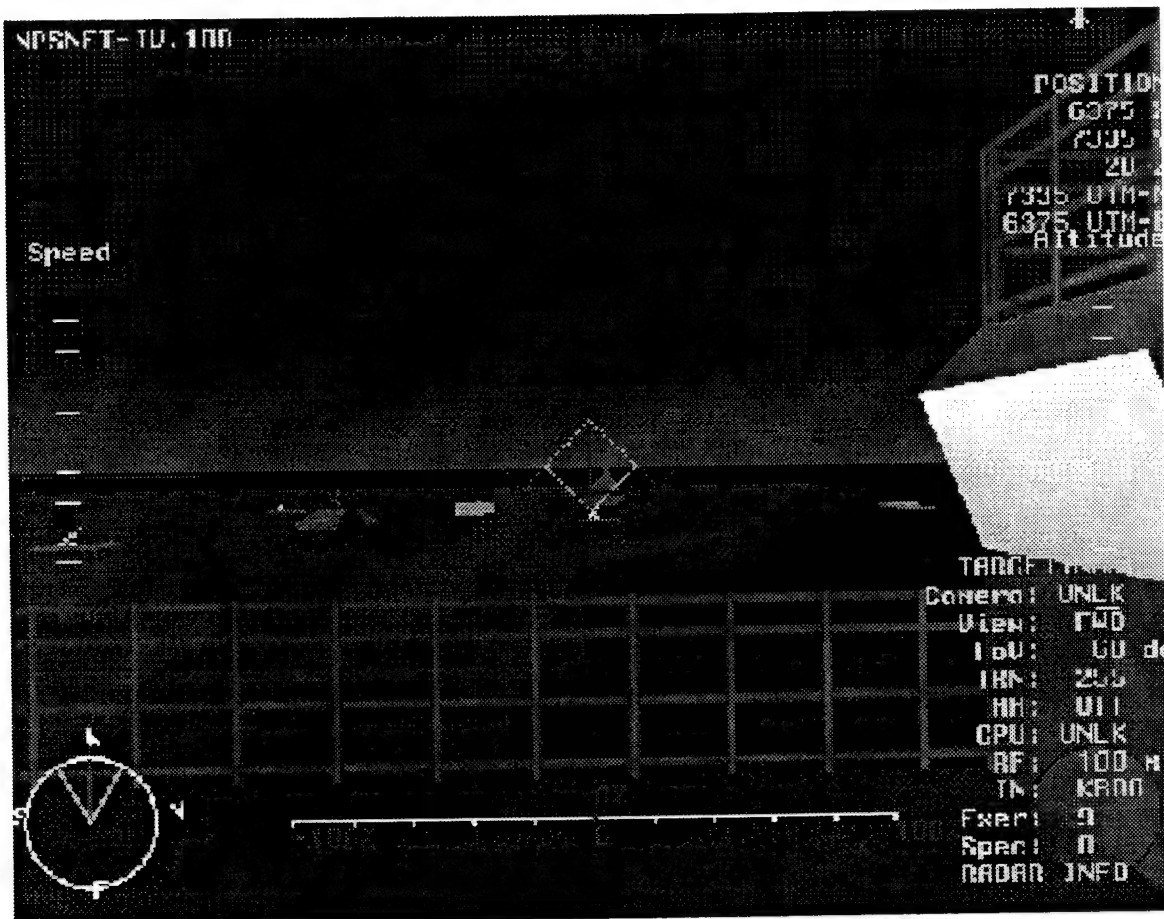


Figure 5: View from the Antares' bridge wing.

C. MOUNTED NETWORKED HUMAN ENTITIES

This work combined DC VET and SHIPSIM to produce a single real-time, networked, interactive virtual environment for shipboard and shiphandling training. Along the way several changes needed to be made to fully integrate the two simulations into NPSSET. These included modifying the PVS algorithm and the addition of an additional network port to allow mounting the human entities.

One problem with DC VET's implementation of PVC is that it requires dynamically changing the database by the algorithm. When a cell is visible it is added to the scene graph and visa versa. However, in NPSNET vehicles are added to the scene graph when it is created. Once created, its database nodes cannot be changed. To get around this, a Performer pfSwitch node is added to the highest database node of each cell [STEW96]. When the pfSwitch is "ON" the cell is sent to the cull and render process as normal. When the pfSwitch is 'OFF' the cell is ignored. In this way the swapping of cells is achieved without the costly dynamic restructuring of the scene graph.

In order to mount the human entity on another (ship) entity over a network, a method of reporting one entity's position relative to the other must be implemented. The current DIS standard does not support such ability. Therefore a separate network, HIRESNET, was created for this purpose and to pass more detailed information about the human entity such as upper body and arm joint articulation. Detail of this and other possible solutions will be discussed in a later chapter.

The major accomplishment here is the ability to allow human entities to successfully mount a ship entity, move around and interact with that ship as it negotiates its own virtual environment. The concept is central to this research and is expanded to accomplish this thesis.

III. SOFTWARE ARCHITECTURE

A. BACKGROUND

Amphibious and littoral operations are among the most complex, dynamic, and manpower intensive evolutions in the Navy. Their simulation involves a large number of entities, both human and vehicular, and place an enormous load on the computers used. At the same time, however, realistic training with these simulations demands fast frame rates. Also, the ever changing types of operations involved requires the ability to easily add new capabilities/doctrines as they are developed. As a result there were two significant design objectives for the amphibious simulator; real time rendering and modularity (or extensibility).

1. High Speed Rendering

The amphibious simulator creates a virtual environment reproducing landing craft and docking ships in rich detail. In addition, the users of this simulation require a high degree of realistic interaction between participating entities to obtain valid training. The high polygon count of the LPD and LCAC models (approximately 50,000) adversely effect the simulation's rendering speed or frame rate. Real time simulations require a minimum of six frames per second for an interactive system with smooth motion beginning at about fifteen frames per second [PRAT93]. The simulation of ships and landing craft moving through the water and docking with one another require sufficient frame rate to show continuous motion of the vessels and adequate response times to maneuvering input. To achieve these

frame rates, an SGI Reality Engine² (RE²) workstation and the IRIS Performer API libraries were used.

2. Additional Functionality

The second objective for the amphibious simulator was modularity. The need, in this case, stemmed from reasons other than simple adherence to software engineering principles. Ease of future expansion is one of the goals of this research since it only supports some of the basic functions required in amphibious operations. Adding new capabilities and vehicles to the simulation must be relatively simple and not require completely rewriting, or even seeing, all previous work. A building block approach can now be used to plug in additional functionality, via modules, as they are developed. Modular design, achieved through the help of tool kits such as EasyScene 3.0, creates the potential for such growth.

3. NEXTGEN and NPSNET V

A further case for modularity is that it facilitates porting from one API to another. It is fully intended that this work be incorporated into NPSNET V.

NPSNET, conceived in 1990, has evolved, through several iterations, to its present form, NPSNET IV. Its complex nature and long history has rendered it extremely difficult to maintain and complicated to extend, and each additional modification compounds the problem. To reverse this trend, NPSNET V will be a virtual environment built completely from a general purpose toolkit, NextGen, currently being architected at the Naval Postgraduate School (NPS).

NextGen is a virtual environment toolkit for both the SGI and PC platforms. It will be available as both C++ libraries and be DIS/HLA compliant. NPSNET V's framework will be built up, piece by piece, from student developed modules written with NextGen. Each piece is written

independently from the others, yet all work together; much the same way the different parts of the amphibious simulator are brought together.

B. IRIS PERFORMER API

IRIS Performer was developed for creating real-time, three dimensional graphics and visual simulation applications [SGI94]. Applications using Performer's libraries take advantage of its native ability to optimize performance on any SGI system. Performer also has two other major features which make it most appealing to this work.

1. Multiprocessing

Performer's multiprocessing capability divides an application into three different processes; the simulation or application process, the culling process, and the draw process. (An optional fourth process, the intersection process, is not covered here, but will be discussed in greater detail in a later chapter.)

The application process is entirely responsible for conducting required computations such as updating entity state (position, velocity, orientation), hydrodynamic equations, etc. It also handles user inputs and network traffic. Application functions, therefore, do not always occur prior to the cull and draw processes. To accommodate this, it is further split into two other processes, the pre-frame process which occurs prior to the cull and draw, and the post-frame which occurs afterward. The culling process conducts a recursive search of the application's scene database to determine which nodes are within the viewing frustum. If so, those nodes are added to the display list and later rendered by the draw process. Those nodes, and all their children, which are not within the viewing frustum are pruned from the

scene graph. Performer's run-time loop then, continuously cycles through these processes until the program is terminated (Figure 6).

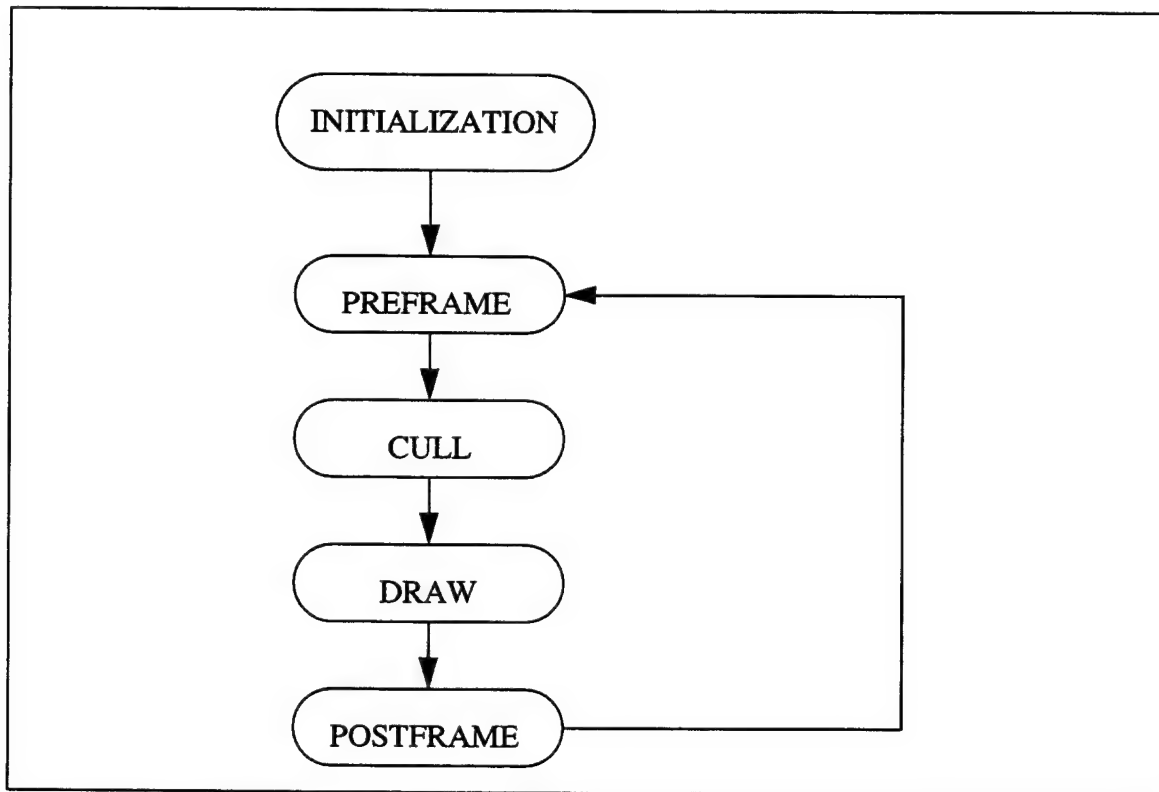


Figure 6: Performer run-time loop.

The advantages of this arrangement become obvious on multiprocessor workstations such as the RE². Each process can be assigned to a different processor, thereby greatly increasing the overall frame rate. In this best case scenario, the processes are pipelined across the different CPUs (Figure 7). At a given time, one CPU draws frame (n), another CPU culls frame (n-1) and a third handles the application process for frame (n-2). In this way, three different frames, each in a different stage of completion, are worked on at once with a theoretically possible three-fold performance increase. Any information which is needed across processes is stored in a shared memory data pool (pfDataPool).

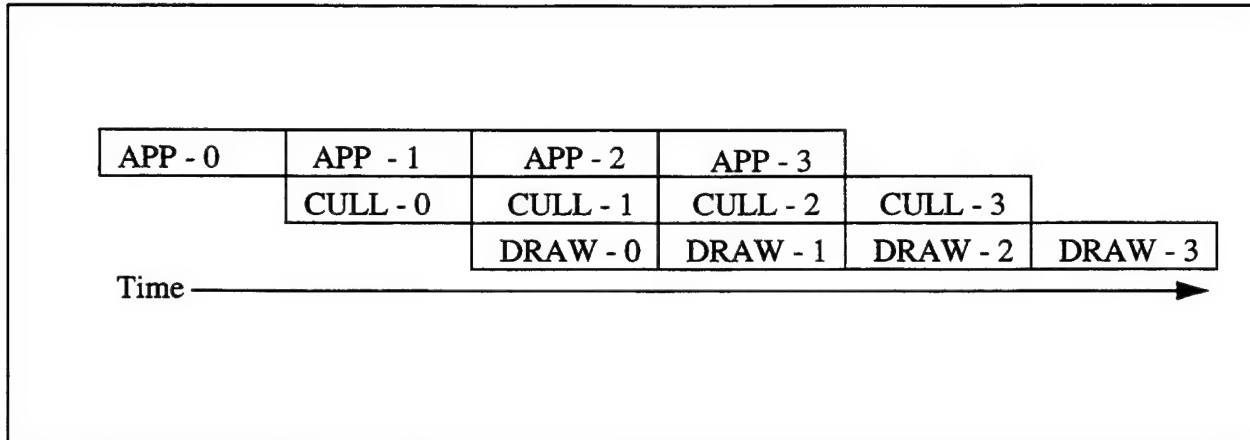


Figure 7: Performer process pipeline.

2. Hierarchical Database

Another prominent feature of Performer is its scene graph hierarchy. This visual database is a directed, acyclic graph (DAG), or tree, of various types of nodes rooted at a scene node (pfScene). The scene is rendered through one or more channels (pfChannel) which, in turn, are culled and then drawn in the graphics hardware pipe (pfPipe). Other nodes include pfDCS and pfSCS (Performer's dynamic and static coordinate system representations) which control the behavior of objects such as vehicles and terrain respectively. All mobile or articulated objects must be assigned to nodes of type pfDCS. The actual geometry of a model is contained in the leaf, pfGeoset, nodes. A useful feature, pfSwitch nodes, allows turning geometry on and off. This removes them from culling and drawing consideration.

The scene hierarchy is created starting at the scene or root node with child nodes added according to some logical or spacial organization. A simplified version of the amphibious simulator's database is shown in Figure 8 as an example.

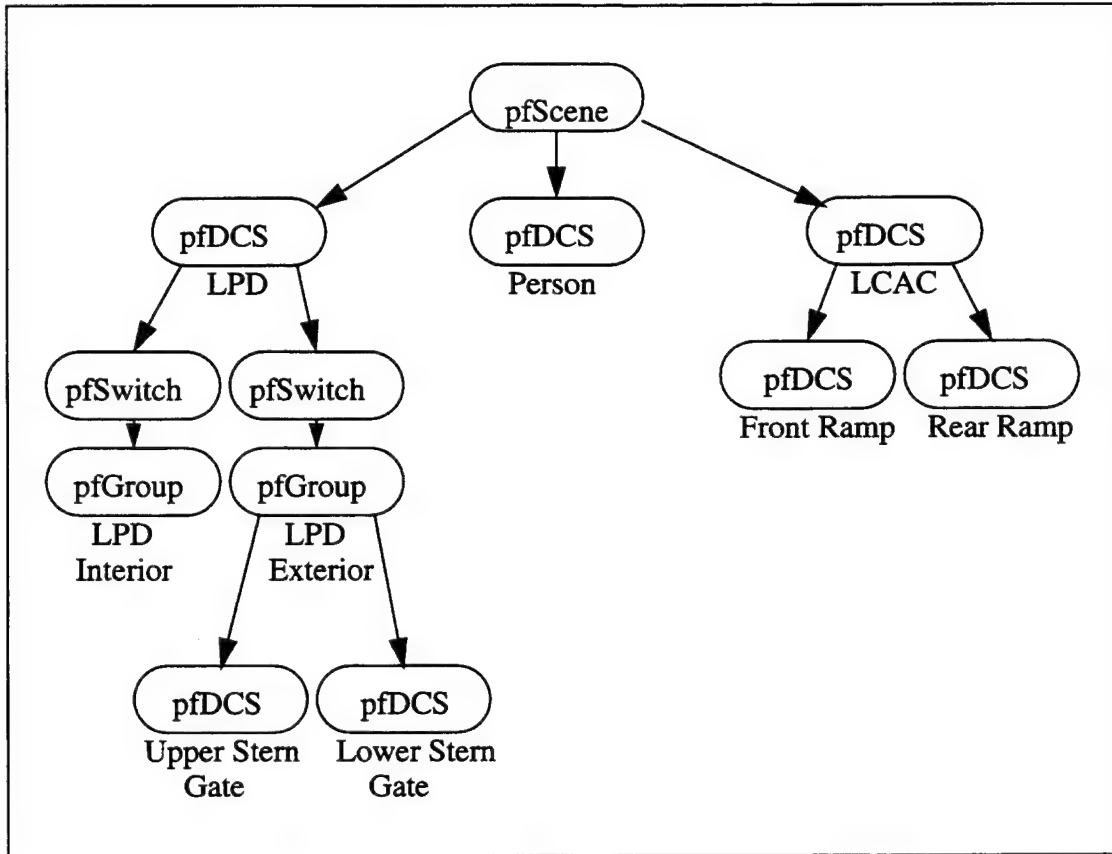


Figure 8: Simplified scene graph for the amphibious simulator.

Inheritance is one of the major benefits of this type of hierarchical structure. In Performer all attributes of a node are inherited by its children. This includes such things as material and lighting properties. Switch values also propagate downward; i.e. if a parent node is not drawn neither will any of its children.

The most critical effect of inheritance is on the pfDCS and pfSCS nodes. Any coordinate system assigned as a child to another is considered local to its parent. Any changes to a child's position or orientation are applied in reference to the parents transformation matrix. Taking the LCAC model's structure as an example (Figure 8), the top level pfDCS node is assigned directly to the scene and its transformations are in the world coordinate system. The loading ramp nodes, however, were made children of the top level pfDCS and have their own coordinate system. As a

result, any transformation of the parent node is inherited by the ramp nodes (they move with the rest of the LCAC), but the ramps are capable of independent movement (raising and lowering themselves).

C. EASYSCENE 3.0 API

EasyScene 3.0 (es3.0) is a real-time, three dimensional, development toolkit built on top of the Performer API [CSIA96]. It is, in fact, a high level abstraction of Performer's more primitive functionalities and, as such, all the previously mentioned advantages and capabilities of Performer apply. The only significant difference between the two pertain to coordinate systems (Table 2). Both are right-handed coordinate systems but differ by a 90 degree rotation about the x-axis.

Table 2: EasyScene to Performer coordinate conversions.

EasyScene/ Open GL	Performer
x-axis	x-axis
y-axis	+ z-axis
z-axis	- y-axis
h (heading)	h (heading)
p (pitch)	p (pitch)
r (roll)	- r (negative roll)

1. Structure

Through this encapsulation, the es3.0 API provides the benefit of greatly reducing the amount of code needed to accomplish a task. However, no high level API can cover all

requirements. To overcome this, es3.0 allows access to the underlying Performer nodes. This also assures that the user always has low level control of the application whenever needed.

An example of this simplification is the esObject, one of es3.0's most basic building blocks. The amphibious simulator contains numerous esObjects, such as dynamic objects (vehicles) and interactive objects (cameras). The esObject structure encapsulates geometry and various parameters for an entity in the visual simulation. This is the smallest homogenous unit for containing and operating on geometry in EasyScene. Each esObject contains the following:

- Name
- Position (x, y, z)
- Orientation (heading, pitch, roll)
- Rotation/translation matrix
- Velocity
- Bounding volume information
- Visibility information
- Scene information
- Collision data
- Pointers to this object's parent and children
- Handle to all Performer nodes used, such as pfDCS, pfSCS, and pfGroup
- Anonymous structure for user defined data

Figure 9 demonstrates how to instantiate and manipulate an esObject. Line 1 loads the geometry contained in the file "lcac.flt", and adds the new object to a linked list of other esObjects in the scene graph. Line 2 translates the object to the position specified by x, y, z and line 3 re-orientes the object. All translations and rotations are absolute and not

```

{
1. esObject *obj = esAddObj("lcac.flr", scene);
   ....
2. esSetObjectPosition(obj, x, y, z);
3. esSetObjectOrientation(obj, heading, pitch, roll);
   ....
}

```

Figure 9: esObject manipulations.

cumulative. After calls such as in lines 2 and 3 are made, the esObject structure described above is automatically updated and the data is available to the user.

By contrast, gathering the same information directly from Performer would entail significant work (albeit more tedious than difficult in most cases). For example, in order to extract the same position (x, y, z) and orientation (heading, pitch, roll) data which is automatically provided by the esObject, the user must first obtain the pfDCS nodes translation matrix (via pfGetMat()). Secondly, the position and orientation values need to be derived from the matrix (not trivial). Repeat every frame.

Other es3.0 basic object types are esChannel, esPipe, and esScene. esChannels can be attached to any esObject in the scene. Once attached to an object, as the object moves so does the viewpoint. In addition, an offset can be specified which moves the viewpoint from the origin of the attached object. Attachment and detachment of esChannel can be performed by the application at any time. The es3.0 notion of a camera is simply an esObject with no associated geometry attached to a channel. This provides a convenient viewpoint into the application. The amphibious simulator's implementation of the "personClass" is based, in part, on a camera. An esPipe contains information on how many

scenes are to be rendered. In most applications there is only one scene, but if multiple hardware pipes are available then several esPipes may be displayed at once. An esScene is equivalent to Performer's pfScene and contains pointers to a pfScene and to a linked list of esObjects contained in the scene.

2. Run-time Interface

The last fundamental es3.0 object is the esModule. As its name implies, it is at the heart of EasyScene's modular capability. An esModule is a container class for a set of user defined callback functions which are executed at run-time. It gives a predetermined entry point into EasyScene's run-time cycle, which is closely linked to its Performer counterpart. This run-time cycle pre-exists within es3.0. The user needs only to define the modules which interface with it. Table 3 lists the different possible callbacks available to each module and Figure 10 shows how the most important of these interface with Performer. Not every module needs to implement every callback. In fact, rarely are more than two or three required. The most commonly used include: preinit, which is called prior to the process split and is required if any shared memory arenas are declared; preSim, which is called prior to the start of the frame and is where time critical events should be placed; and preCull which occurs prior to the cull process and is where any customized culling should be accomplished.

Table 3: esModule callbacks

esModule callbacks
void (*preinit) (void)
void (*init) (void)
void (*postinit) (void)
void (*preSim) (void)

Table 3: esModule callbacks

esModule callbacks
void (*postSim) (void)
void (*preCull) (void)
void (*postCull) (void)
void (*preRender) (void)
void (*postRender) (void)
void (*exit) (void)

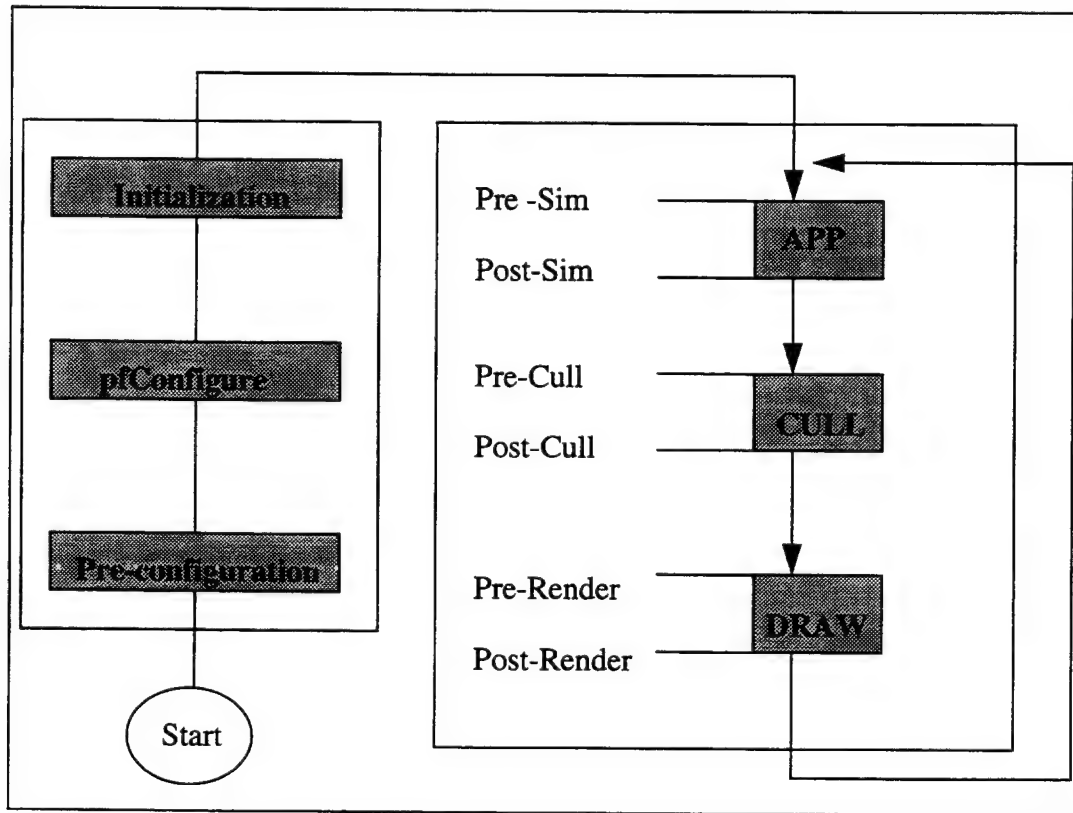


Figure 10: EasyScene interface with the Performer run-time loop. After [CSIB96].

The power of the esModule is its ability to conveniently group together functions with a similar purpose. An example is the amphibious simulator's "person_module". All

functionality for each instance of the “personClass” is handled through this module, ensuring that the proper member functions are invoked at the correct time in the run-time loop. The “person_module” contains three callbacks: “init_dude”, an init callback, simply instantiates the person object; “update_dude”, a preSim callback, takes care of calling the member functions which move and otherwise update the person object’s state every frame; lastly “cull_dude”, a preCull function, which does simple visibility culling based on the object’s location.

D. SUMMARY

The amphibious simulator was designed with real time performance as well as modularity in mind. This is accomplished through the help of the Performer and EasyScene toolkits. The run-time behavior of the simulation is controlled by a set of self contained modules (Table 4) which represent the different capabilities of the system. The “time_module” provides the time since last frame data (delta_time) required by the vehicle’s dynamics model. The “device module” creates and updates the simple keyboard and mouse inputs which can be used to control the simulation. The “ocean_module” provides the realistic ocean wave and ship response models so critical to accurate landing craft simulations. Lastly, the “person_module” and the “ship_module” invoke the “shipClass” and “personClass” (Figure 11) member functions which perform the bulk of the work in the simulation. Each of these modules, as befits the name, can be taken out and replaced by others of similar capabilities, or completely new ones may be added.

Table 4: Amphibious simulator's module structure

esModule	callbacks
time_module	preinit - init_clock preSim - update_clock
ocean_module	init - init_sea preSim - update_sea
device_module	init - init_devices preSim - update_devices
ship_module	init - init_ship preSim - update_ship
person_module	init - init_dude preSim - update_dude preCull - cull_dude

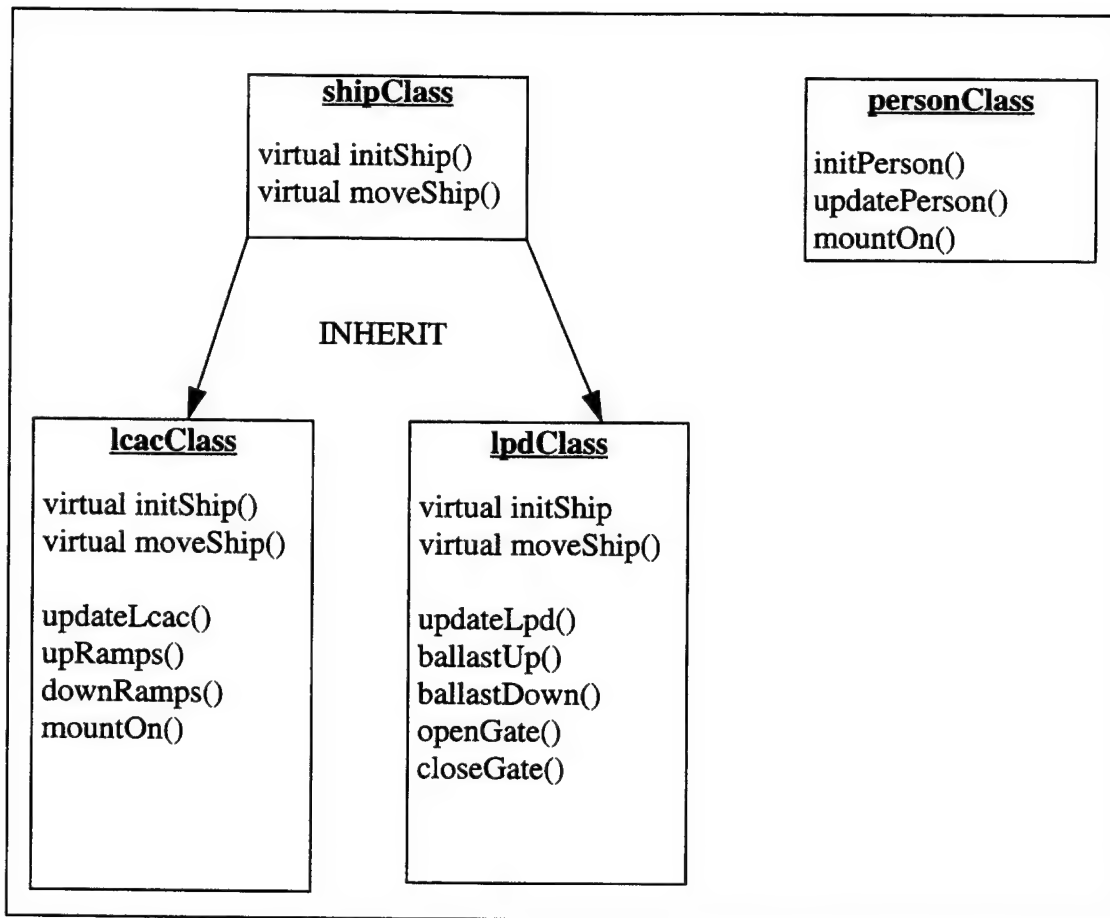


Figure 11: High level class hierarchy for the amphibious simulator.

IV. EASYSCENE 3.0 MARITIME MODULE

A. INTRODUCTION

By their very nature, amphibious operations are at the mercy of unpredictable ocean conditions. For safety reasons, every landing craft in current use is operationally limited by sea state constraints (some more than others). The well deck of large mother ships, such as the LPD-17 used in this work, always runs the risk of mishap when operating landing craft in heavy seas. Even aircraft are not immune, as their pitch and roll envelopes must be respected as well. Therefore, this simulation must take the ocean into account in order to provide realistic interactions among its participants. The es3.0 Maritime Module is a plug-in module which allows a real time, dynamic, ocean based on the Pierson-Moskowitz spectrum to be added to the application.

B. OCEAN MODEL

The Maritime Module creates an ocean model. Each ocean model, in turn, is comprised of two parts: a dynamic sea and a static sea [CSIB96]. The dynamic sea, as the name implies, is the part of the ocean which moves. Its motion is determined by the ocean's energy spectrum (discussed below) and sea state. The sea state is a direct measure of the ocean's total energy content at that moment and indicates the ocean wave's significant height. More formally, the significant wave height is defined by [PNA67] as the average of the highest one-third of all waves as measured over a period of time. The Maritime Module assigns initial significant wave height values to each sea state (Table 5).

The dynamic sea contains wave components that simulate the effects of a rolling sea. This part of the ocean model dynamically renders three dimensional waves with user

Table 5: Marine Module Sea State Defaults

Sea State	Initial Significant Wave Height
1	1 meter
2	2 meters
3	3 meters
4	4 meters
5	5 meters
6	6 meters

definable parameters. The generation of the dynamic sea comes at great computational expense. In order to save computational resources for other parts of the application, it is useful to limit the dynamic sea to the current area of interest - the user's view point. In the case of the amphibious simulator, the dynamic sea is centered about the human entity and follows from vehicle to vehicle. Thus the user's view of the world are always effected by the ocean conditions (Figure 12).

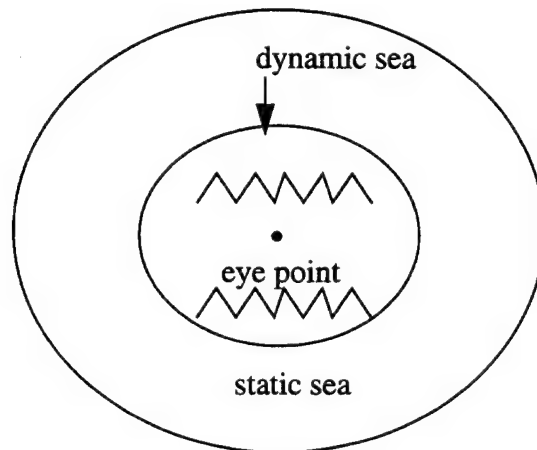


Figure 12: Maritime Module Ocean Model.

The static sea is that part of the ocean which does not move, but rather blends in with the dynamic sea. Its purpose is to visually extend the ocean out to the horizon (or far clipping plane), while conserving computational power. The user determines the extent of both static and dynamic seas based on the trade off between a need for realism and computational cost. Table 6 shows the cost of different size dynamic seas on this simulation. A 225 meter sea diameter was chosen since it provided adequate ocean coverage and still allowed real time frame rates. As can be seen, the performance drops off quickly as the sea radius increases. A radius of more than 1000 meters exceeds the resources of a four processor RE² and results in a segmentation fault.

Table 6: Dynamic sea / performance trade off.

dynamic sea radius	frame rate
225 meters	10 fps
375 meters	7.5 fps
525 meters	5.5 fps
750 meters	4.6 fps
>1000 meters	N/A

C. OCEAN SPECTRA

How does the dynamic sea simulate the behavior of the open ocean? The concept, oceanographers realized in the 1960's, is to represent the irregular sea surface by summing very large numbers of small amplitude sine waves, each of different periods, amplitudes and directions. The phase relationships among these various sinusoids is random. In this

way, any seaway can be characterized by an 'energy spectrum' which indicates the relative importance of each component wave making up the observed, complex pattern [PNA67]. In other words, where the sea state gives an indication of the total amount of energy in the sea, an ocean's spectrum describes the statistical distribution of that energy among the constituent sine waves.

These component waves do not really exist, and so they can not be seen or measured. However, the energy spectrum defining these components can be obtained from the Fourier analysis of observed ocean recordings.

One such recording, and its associated energy spectrum, is the Pierson-Moskowitz spectrum. The Pierson-Moskowitz spectrum is based upon a deep sea empirical study of the North Atlantic ocean. This spectrum assumes that all the ocean energy is coming from one direction, resulting in long crested waves of the type normally associated with deep ocean channels. The source of this energy are far off ocean storms. The Maritime Module also allows for short crested waves, or choppy seas, by distributing the energy contained in the single long crested wave over several direction. The result is several long crested waves, of lesser energy, starting from different points. The mutual interference of these different waves accurately imitates a confused sea.

D. esSea CONTRUCTION

An esSea object is the Maritime Model's implementation of the Pierson-Moskowitz spectrum. An esSea is made up of zero or more esWaves, each traveling in a different direction. A sea with no waves is perfectly flat; one wave results in an ocean with long crested properties; several waves cause the confused state described above (Figure 13).

The esWaves themselves can only have one heading and are comprised of zero or more frequencies (esFreq). Obviously a wave with no constituent frequencies adds nothing to the overall sea. The different esFreq's determine the nature of the wave. An esWave with a single esFreq is

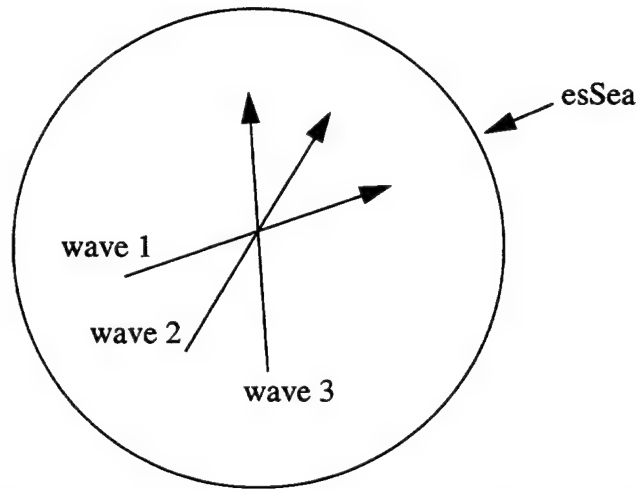


Figure 13: esSea and constituent waves. After [CSIB96].

very regular (just a standard sine wave), two or more esFreq's add increasing unpredictably to the wave. In general, higher frequencies produce the effects of locally generated wind waves, while lower frequencies represent the effects of distant storms.

E. SUMMARY

Adding the ocean model, or any other new capability, to the simulation involves implementing a new module; in this case called "ocean_module". The basic initialization of the ocean is straight forward and is derived directly from previous discussion (Figure 14). Line 1 creates the ocean model with the given dynamic and static sea dimensions. Line 2 instantiates the new waves to which are added esFreq's (line 3). Finally the separate esWaves are put together into the esSea.

```
{  
1. sea = esMarNewSea(CELL_LENGTH, DYNAMIC_CELLS, STAT_CELLS);  
2. for (curr_wave=0; curr_wave<W; curr_wave++)  
   {  
       wave = esMarNewWave();  
3.   for (curr_freq=0; curr_freq<F; curr_freq++)  
       {  
           esMarAddFreq(wave, esMarNewFreq());  
       }  
       esMarAddWave(sea, wave);  
   }  
}
```

Figure 14: Creation of an ocean with W waves and F frequencies.

V. COLLISION DETECTION

Much of the suspension of disbelief critical to training in virtual environments depends on the life-like responses of its simulations. As soon as a person walks through a wall, for example, the illusion of reality is shattered. To achieve these types of responses, real time collision detection is necessary.

Intersection testing is conducted throughout the amphibious simulator. It is used to detect collisions and enforce realistic behaviors among its mobile objects (human, LCAC and LPD).

A. INTERSECTION TESTING

1. Basic Mechanics

Intersection testing consists of checking whether a ray, or intersection segment (Iseg), crosses through any geometry in the scene; if so, a collision is said to occur. In Performer, related Isegs are encapsulated into a structure called a pfSegSet which contains the position, direction and length for up to 32 Isegs. The pfSegSet also holds data common to all Isegs in the structure, of which the most important is its intersection mask (discussed later in this chapter). The pfSegSet can then be associated with an object and together they are considered a 'collider' (Figure 15).

Functionally speaking, colliders are the basic building blocks for all collision detection in the amphibious simulator. All moving objects must be colliders if they are to interact with the other objects they come into contact with. All objects, of any type, are tested for intersection against these colliders and therefore, collisions can only occur between colliders or between a collider and some other object. As shown in Figure 16, the cube collider collides with the wall object only when one of the cube's Isects (intersection rays)

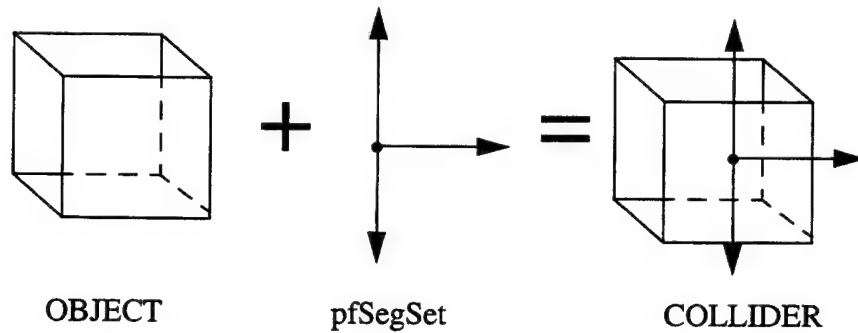


Figure 15: Collider example.

intersects the wall geometry; the overlapping of geometry does not, by itself, constitute a collision.

The scene depicted in Figure 17, in contrast, is not sensed by the application as a collision since

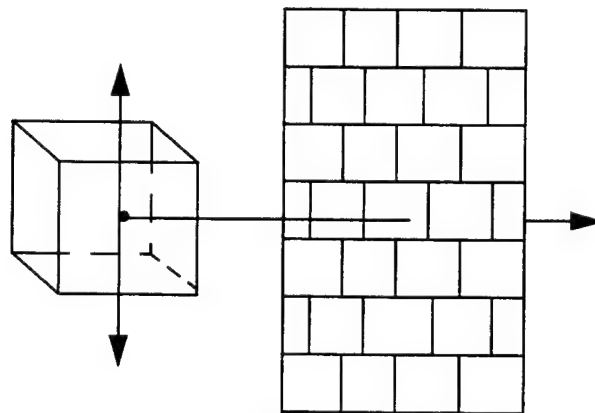


Figure 16: Cube colliding with a wall object.

none of the collider's Isegs intersect the wall's geometry and the wall, a non-collider, cannot initiate any intersection testing.

Once a collision has been registered with the application, a user defined call-back function is invoked to handle it. This call-back is passed a data structure (called an esIsectInfo object in EasyScene) which contains a pointer back to the collider itself, and for each Iseg in the collider:

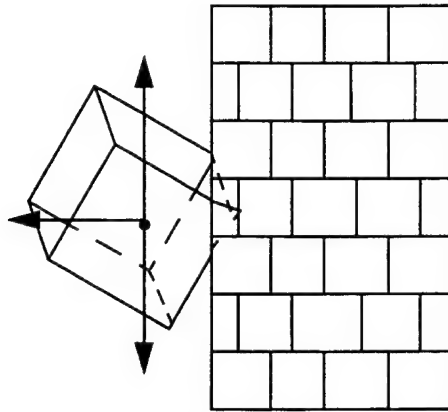


Figure 17: Cube and wall are not colliding.

- the number of objects hit
- a pointer to each object hit
- the collision point
- the normal at the collision point
- the clipped Iseg (that part of the Iseg between the collider and the object hit)

It is now a matter of looping through the different Isegs and extracting the desired information. In general, one such function is needed for each different type of collider. This is where the bulk of the work is done and where the collider's behaviors are defined.

2. Run Time

If collision detection is enabled, once per frame, during the intersection process (this process is optional and not needed if your application does no intersection testing), the scene graph is traversed once again to see if any collisions have occurred since the previous frame. To accomplish this, each Iseg is tested against the geometry of the scene graph. It would be a waste of computational power, however, to test each Iseg against each polygon every frame, since the odds of collision, usually, are statistically very low. Instead, a check between the Iseg and each object's bounding volume is made first. If, and only if, the

object's bounding volume is intersected by the Iseg, a more detailed search of each polygon in that object is made to find the specific intersection data. Since an object's children are also contained within its bounding volume, this method results in a large reduction in required testing. Using the amphibious simulator's LPD and LCAC as an example, if the LCAC is not within the bounding volume of the LPD (as is often the case when it is not mounted) the application does not need to check any of the LPD's 50,000 polygons for intersection with the LCAC's Isegs - a huge saving. The overall effectiveness of this technique is demonstrated by the fact that the use of collision detection has had no degrading effect on the run time performance of this simulation.

B. INTERSECTION AND COLLISION MASKS

Every polygon in the scene has an attribute called a collision mask. A collision mask is a 32-bit bit field (usually represented by a eight digit hexadecimal number) which is used by the intersection process to determine if that polygon has been hit. A default value of 0xffffffff (all 1's) is assigned to each object as it is brought into an application. Like any other attribute, the mask data is inherited by all of that object's child nodes and so each individual polygon will initially contain the default mask value. These values can and will be individually changed to suit our needs.

Every pfSegSet also contains a similar bit field called the intersection mask (mentioned earlier). The value of this field must be specifically set in the pfSegSet structure. The intersection and collision masks are used in conjunction to determine whether or not a collision has occurred.

Collision detection, then, is conducted in three basic steps by the Performer library. The first two have already been discussed at length, and consist of testing each Iseg against the bounding volumes of all objects in the scene and then, if successful, against that object's actual polygons. If, in step two, an Iseg is found to intersect a polygon, the third (and final) step is to conduct a bit-wise AND operation on the Iseg's intersection mask and the polygon's collision mask. Any non-zero

result indicates a collision (Figure 18). The default collision mask will always test true in

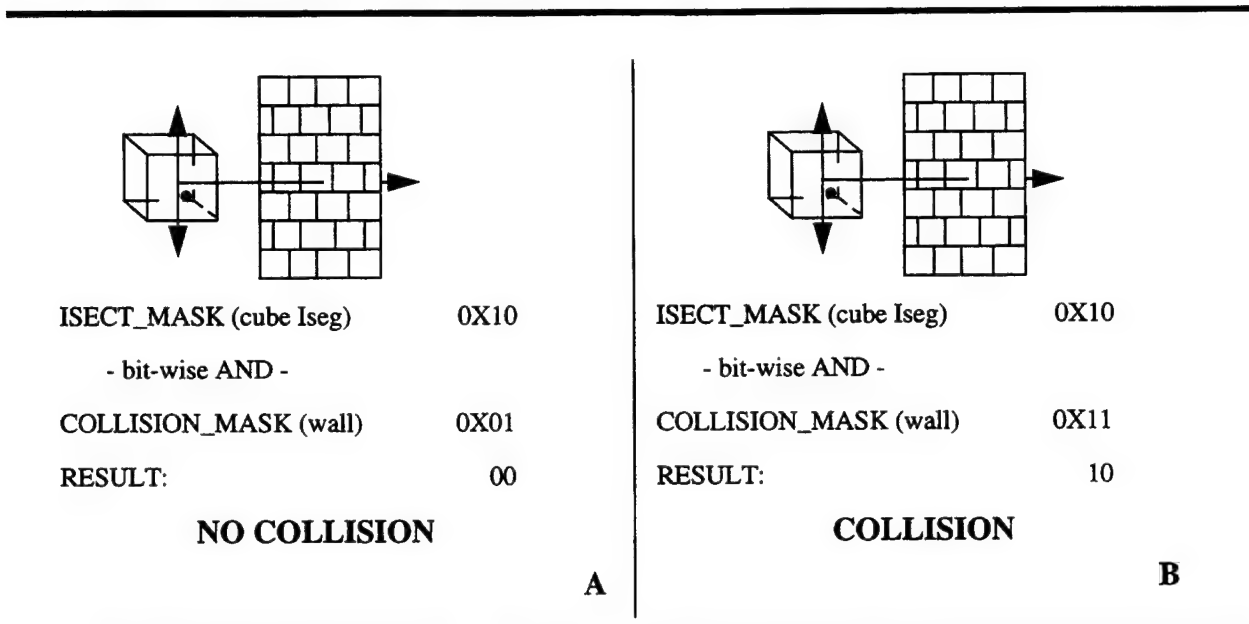


Figure 18: Collisions based on intersection and collision masks.

this last step.

This third step lets the user selectively decide how (or if) an object will interact with another. By carefully choosing the values of the two masks, it is possible to make a ship float on water but a person sink, etc.

C. IMPLEMENTATION

Each of the three central object types in the amphibious simulator (human, LCAC, and LPD) rely heavily on collision detection for their behaviors. In all cases, Isegs were constructed around the entities to act as sensors, watching out for intersections with other objects. Some of these sensors were used to avoid collisions while others actually require collision to invoke their underlying behaviors.

1. LPD

The LPD was the simplest of the objects to control in case of collision. Its mass compared to the other vehicle (LCAC) is so great that a collision between the two has no appreciable affect on the LPD and no special behaviors were needed. The only requirement was that it not be able to pass through the LCAC and this was easily taken care of by placing two bands of Isegs around the hull at different heights to sense any collisions which may occur.

One last modification needed to be made to this model. The LPD is physically broken up into two parts, LPD interior and LPD exterior (Figure 8). In order to allow mounted entities to tell the difference between the two, each part was assigned a mutually exclusive collision mask, LPD_INTERIOR_MASK and LPD_EXTERIOR_MASK respectively. Now, an intersection mask can be devised which will react to the inside but not the outside of the model (the converse also holds). Additionally, a bit-wise OR of these two masks was assigned as the LPD_MASK and will allow other entities to test if any part of the LPD is encountered.

2. LCAC

The LCAC uses collision detection for two important reasons; first to handle collisions at sea and secondly, to properly conduct embarkation/debarkation evolutions. To respond correctly to a collision at sea, the LCAC has been logically partitioned into four zones (made up of eight Isegs). The direction in which the LCAC will bounce in reaction to a collision depends on which zone was hit. The resultant twist about the vehicle's pivot point is illustrated by the direction arrows in Figure 19.

In order for the LCAC to embark on the LPD, it must first recognize that it is in the LPD's well-deck. Therefore, two additional Isegs were added to the LCAC, one at the pivot point and the other at the stern, both pointed down. When both these Isegs sense a LPD_INTERIOR_MASK,

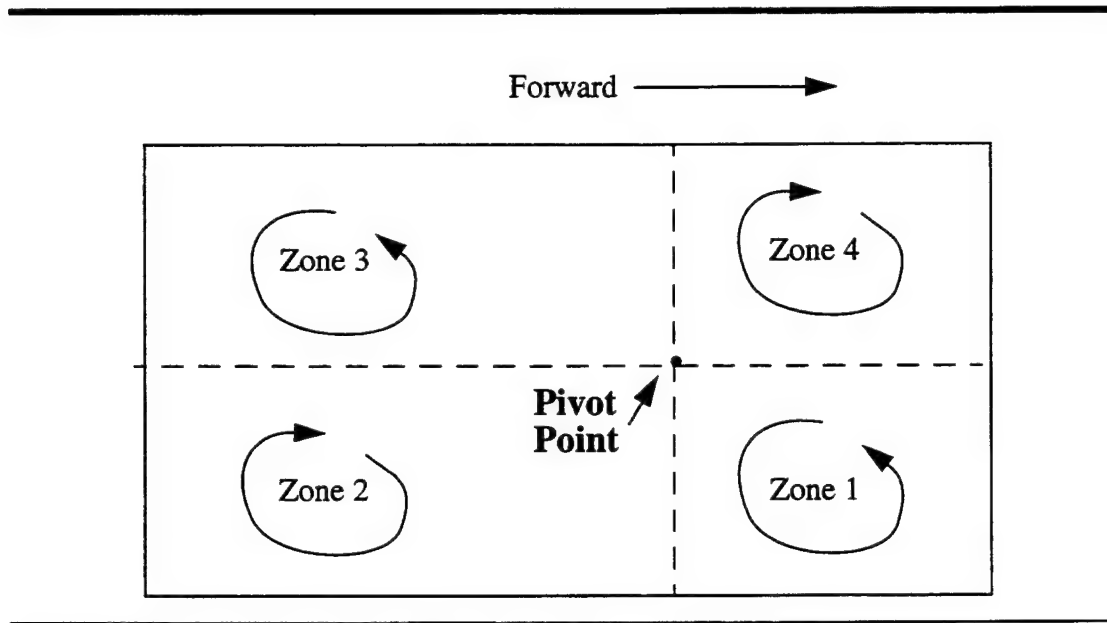


Figure 19: LCAC collision zones.

the LCAC is completely inside the well-deck and the mounting algorithm is applied (examined in detail in the next chapter). When the stern Iseg is no longer in contact with the well-deck, the LCAC is backing out, and is once again treated as an independent vehicle. These Isegs only react to the geometry within the well-deck, so mounting can only occur in the proper situation, i.e. when the LCAC is completely within the LPD's well-deck and not by random collisions with any other object.

3. Human

The human entity used in the amphibious simulator is a disembodied representation of a typical sailor. In order to walk around the LPD and LCAC in a realistic manner, he/she must: occupy space, climb ladders (staircases) and ramps, and be confined by walls and other solid objects. Toward this end, the person object is physically defined by a group of Isegs of roughly human proportions (Figure 20). The person is prohibited from walking

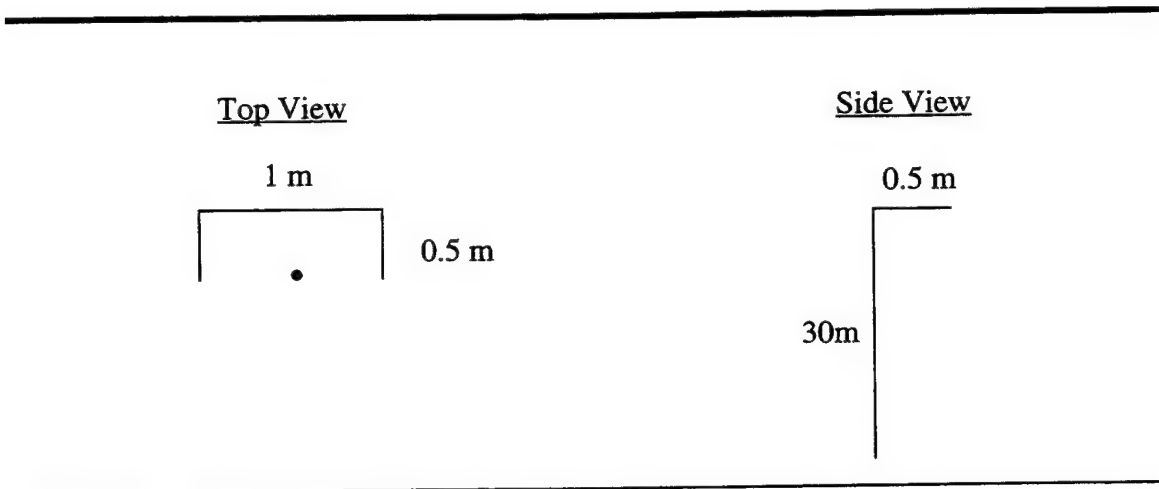


Figure 20: Human entity dimensions.

through walls, pillars, and other obstacles by the forward facing group of three Isegs surrounding its head. These Isegs are 'cooked', meaning that they re-orient themselves with the person; always facing in the direction the person is moving. The person's intersection mask is set to 0xffffffff and therefore will collide against all other objects. In the event of a collision on one of these Isegs, the person's collision call-back function will cause the person to take a step back to its previous, non-colliding, position.

The person's height above ground is maintained by the last Iseg at a constant two meters. The Iseg is aimed straight down to a length of 30 meters. This might seem excessive but is required to ensure the person remains in touch with its parent in all cases. When a ground collision is sensed, the collision point is extracted and the person's height value (y-value) is made to equal the ground's y-value plus two. In this way the person is easily able to walk up ramps and ladders and jump from LCAC to LPD.

Lastly, like the LCAC, the human entity uses collision detection to determine where it is mounted. In the person's case, the decision is easy. If the collision data returns that the object hit is the LCAC, then the mounting algorithm is used with the LCAC as the parent; else, if the LPD is hit, the LPD becomes the parent.

D. CONCLUSION

As evidenced by the examples used in this chapter, this work would not have been possible without the use of collision detection. The ability to allow objects to act, and react, in real time and in a natural way enhances the feel of this simulation. Entites are made aware of the world around them, and therefore, can be more discriminating in their behaviors.

VI. MOUNTING ENTITIES

A. INTRODUCTION

An entity is considered 'mounted' on another entity if its motion is relative to that other, parent, entity. A commuter riding a train typifies this relationship. When the commuter steps onboard, he mounts the train. The person is then carried along with the train, but is also able to move within its confines and to alight when desired. While onboard, the person is considered a 'child' and the train is called the 'parent'.

In order to accomplish its goals of demonstrating the interactions of amphibious vehicles, the amphibious simulator takes this concept a few steps further. The vehicle and human entities need to be able to mount and dismount each other at will (change trains) as the simulation progresses. Additionally, the ability to nest mounted entities is required, e.g. human on LCAC on LPD.

A generic technique for mounting, which allows any logical parent-child combination, was developed; an illogical combination would be the LPD mounting a human, for example. This technique enables the entities to dynamically change parentage at run-time and allows for deeply nested family trees (three or more entities mounted on each other). The most useful of these combinations are: all three entities (LPD, LCAC, and human) acting independently - no parent-child relationships; human entity mounted on LCAC or LPD - a single parent-child pair; LCAC and human independently mounted on LPD - one parent with two children; human mounted on LCAC which in turn is mounted on LPD - a grandparent-parent-child situation. In any of these cases the LPD can act only as a parent entity, the person only as a child, and the LCAC as both parent and child.

B. DIS RESTRICTIONS

As mentioned in previous chapters, it is intended that this work be later incorporated into NPSNET V which will be a DIS/HLA compliant system. To further this goal, every effort was made to ensure that the amphibious simulator produces entity data compatible with the DIS protocol. Position reporting is the most important of these compatibility issues.

In order to maximize performance, DIS relies on the User Datagram Protocol (UDP) for data transmission which is a 'best try' (technically unreliable, but usually good enough) transmission protocol. The advantage of UDP is that it has been shown to be up to ten times faster than its reliable counterpart, Transaction Control Protocol (TCP)[GOSSW94]. The unreliable nature of DIS has two important implications for this work. First, incremental data can not be used to maintain a correct view of the world since the user can never know if he has received all the updates. If an update is missed, there is no way to recover. Future incremental updates assume all previous ones were received and may lead to incorrect positioning. Entity movement information must, therefore, be given by absolute positions. Luckily, it is a trivial matter to have each entity keep track of its own position updates and report it to the application in the required, absolute, form.

The second, and more important, restriction is that DIS does not support local coordinate systems (or nested object hierarchies). Again, since there are no guarantees on packet reception, an application will never know if it misses a message re-assigning an object's position in the hierarchy. Any subsequent update to that object will be made in error. As a result, all entities must reside in the world, or universal, coordinate system (a flat object hierarchy).

C. MOUNTING ALGORITHM

Unfortunately, this second restriction eliminates the most obvious, though not necessarily the best, approach to the mounting problem. Performer (and EasyScene) allows one object node (pfDCS) to be dynamically attached to another. In terms of the matrix stack, this pushes the mounting entity's transformation matrix on top of the mounted entity's matrix. As shown in Figure 21, this has the effect of Object B being local to (or mounted on) Object A.

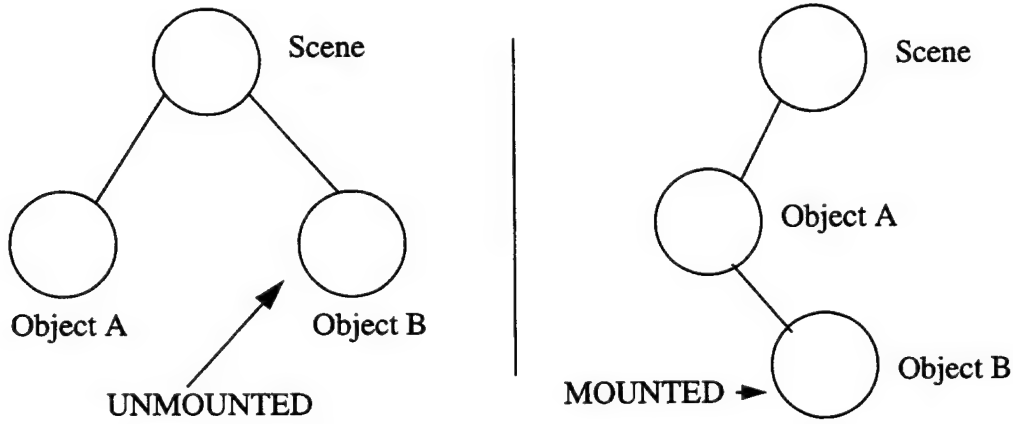


Figure 21: Mounting entities through local coordinate systems.

An alternative algorithm was developed which satisfied both DIS imposed restrictions and proved more flexible than the method described above. This is a generalization and extension of preliminary work done in [STEW96].

The mathematical foundation for the algorithm is based on Equation 6.1 which is adapted from [CRAI89],

$${}^B P = \begin{bmatrix} A \\ R \end{bmatrix} {}^B P_{Local} + {}^A P \quad (\text{Eq 6.1})$$

where ${}^B\mathbf{P}$ is the global position of the mounting entity (B), ${}^A[\mathbf{R}]$ is the rotation matrix of the mounted entity (A), ${}^B\mathbf{P}_{Local}$ is the local position of the mounting entity relative to the mounted entity, and ${}^A\mathbf{P}$ is the global position of the mounted entity.

Take, as an example, the case where an LCAC is embarked on the LPD. Equation 6.1 now reads:

$${}^{LCAC}\mathbf{P} = [{}^{LPD}\mathbf{R}] {}^{LCAC}\mathbf{P}_{Local} + {}^{LPD}\mathbf{P} \quad (\text{Eq 6.2})$$

Figure 22 is a static representation of applying Equation 6.2. The result passed on to the

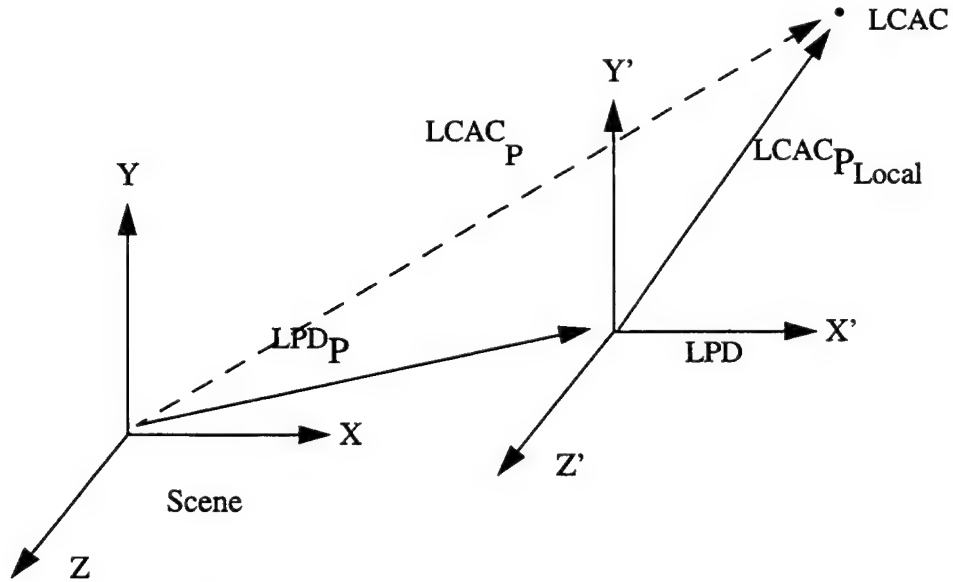


Figure 22: LCAC mounted on LPD.

application is the global position of the LCAC, ${}^{LCAC}\mathbf{P}$. In object-oriented fashion, each entity class is responsible for keeping track of its mounted status and for calculating its own versions of Equation 6.1. In this way all local coordinates are removed and all the main simulation ever sees is global positions.

The more complex case, where there are nested mounted entities, is handled in the same way. A human is now added and mounted on the LCAC in the previous example. The human entity performs its calculations based on its parent's (the LCAC) global position and rotation matrix. This information is provided directly by the parent. The fact that the LCAC itself is mounted on the LPD is immaterial and doesn't affect the human entity's calculations (Figure 23). The LPD's movements effect the LCAC's global position which, in turn, automatically trickles down to the human.

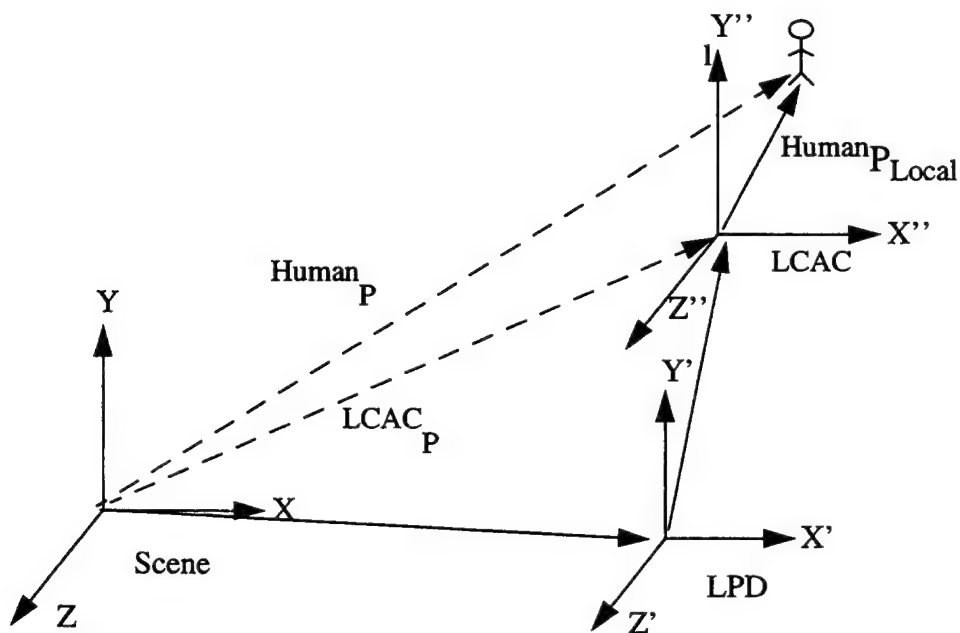


Figure 23: Deeply nested human mounted on LCAC.

These calculations are performed once each frame. As the human walks off the LCAC and onto the LPD, the human entity's intersection callback function (Chapter V) will note the change in parent and now the LPD's position will be used by the mounting algorithm.

The LCAC's movements no longer have any effect on the human entity since it is no longer part of its 'family tree'. In this way entities in the amphibious simulator are free to embark/debark other entities as needed and in any combination.

Pseudo-code for the complete algorithm is as follows:

```

if(MOUNTED)
{
    //Need to determine which vehicle the entity is mounted on,
    // and get the entity's local position on that vehicle.
    switch(mountedOn)
    {
        case LPD:
            parentVehicle = LPD;
            localPosition = getLocalPos();
            break;
        case LCAC:
            parentVehicle = LCAC;
            localPosition = getLocalPos();
            break;
    }

    //Get the parent vehicle's rotation matrix
    rotMat = parentVehicle->getRotMatrix();

    //Apply Equation 6.1
    globalPosition = parentVehicle->globalPosition + (rotMat * localPosition);
}
else if(!MOUNTED)
{
    globalPosition += getUpdatePosition();
}

return (globalPosition);

```

D. CONCLUSION

Dynamically embarking and debarking landing vehicles and human entities, on and off, an amphibious mother ship is central to this work and is the basis for all amphibious operations. The ability to arbitrarily mount entities during a simulation permits training to be conducted in a manner closely resembling the way humans behave in the real world. The algorithm presented in

this chapter allows for the conduct of simple ship-to-shore movements and LPD-17 ship walk through and familiarization. However, it could just as easily be applied to helicopter borne assaults, passenger airliners, trains or any other situation where one or more entities need to operate within another.

VII. MODELS

A. INTRODUCTION

The use of visual simulations for effective training requires highly detailed and accurate models. Inaccuracy is not only counterproductive but could very well turn out to be dangerous. To use an extreme example, a fire fighter previously trained in a ship mock-up might, in a smoky environment, make a wrong decision based on his memory of the model instead of the actual space he is in. Additionally, the realism imparted by a richly detailed model adds much to the immersive nature of the environment. The amphibious simulator makes use of the LPD-17 and LCAC models developed by Advanced Marine, Inc. for Naval Sea Systems Command (NAVSEA).

1. LPD-17

The LPD-17 model is a precise 3-D CAD representation of the proposed LPD-17 class amphibious landing ship constructed using the MultiGen modeling tool. The enormous detail and complexity of the model contribute much to the above mentioned immersion and make the model suitable for a variety of uses such as the walk-through familiarization and well-deck operations demonstrated by this thesis.

This same level of detail, however, is also the model's greatest drawback. The complete model consists of 13 linked files, each describing a different part of the ship (Figure 24). When loaded into an application, they form an object of over 110,000 polygons. This exceeds the capability of even the most sophisticated graphics computers to render and animate in real-time.

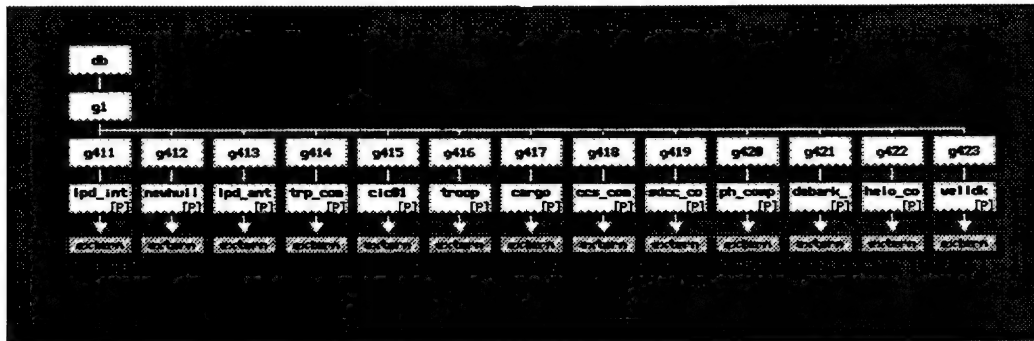


Figure 24: Complete LPD model hierarchy viewed in MultiGen.

Normally, a very effective method for dealing with such a large model is to implement a Potentially Visible Set as described in Chapter 2. However, this would require extensive remodeling and more time than available to complete this project. As a result only part of the model is actually loaded into the application; the outer hull, well-deck and interior structures (Figure 25).

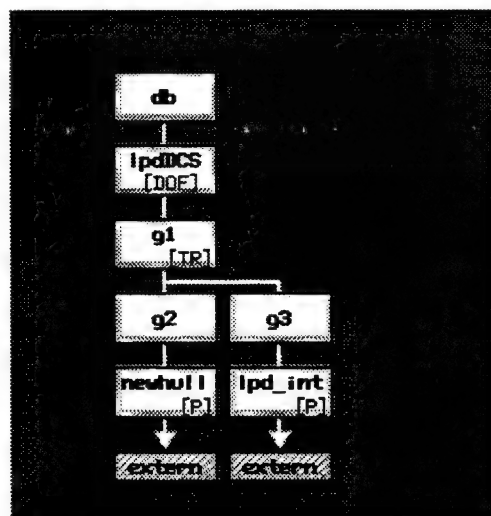


Figure 25: Model hierarchy for amphibious simulator.

Other spaces such as CIC and pilot house are omitted due to their large size. Fortunately they are not necessary to fulfilling the goals of this simulation.

2. Scenes from the Amphibious Simulator

Figures 26-28 show various scenes from the amphibious simulator. Video footage of a simulation in progress is available from the Naval Postgraduate School.

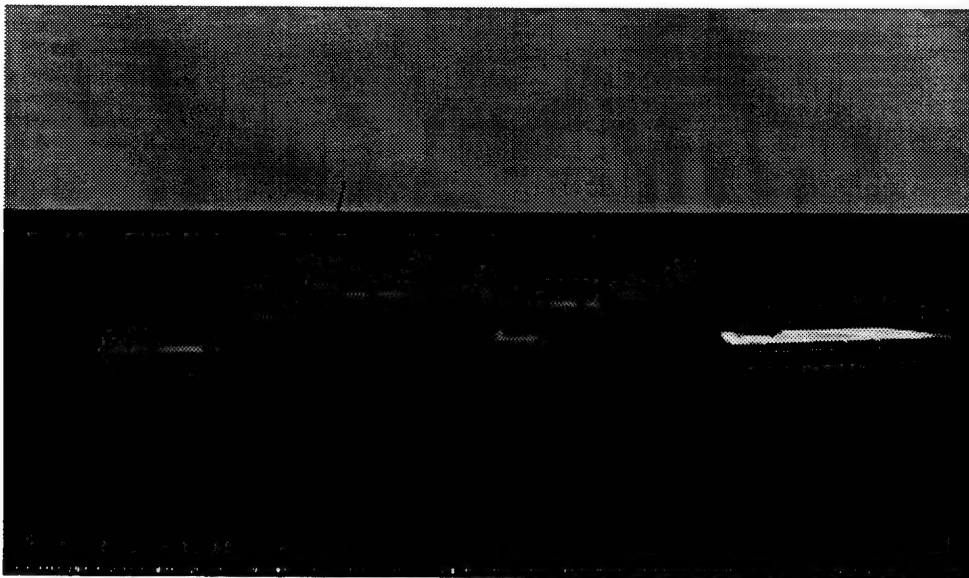


Figure 26: Side view of LPD-17.



Figure 27: Human's view of LCAC in well-deck.

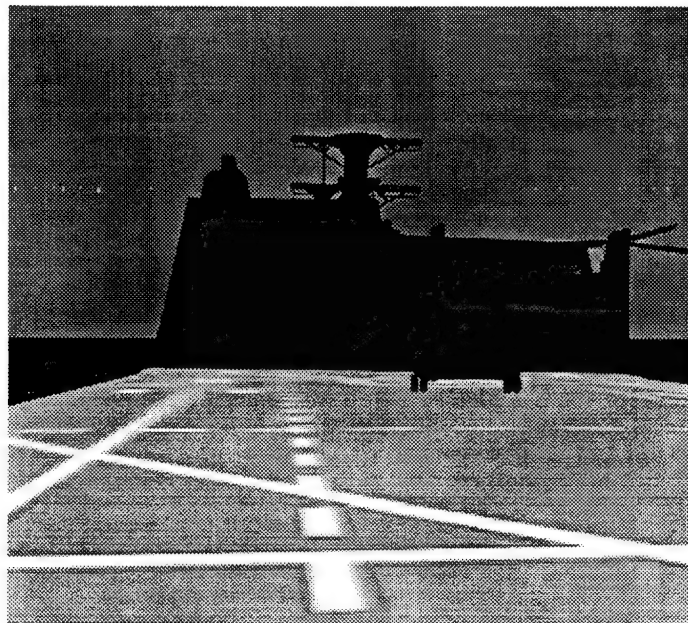


Figure 28: Human on LPD's flight deck.

Lastly, Figure 29 and Figure 30, although not part of this work, are included to document some of the LPD-17's other major spaces as well show that CIC, in particular, is well enough developed to support an extensive simulation of its own.

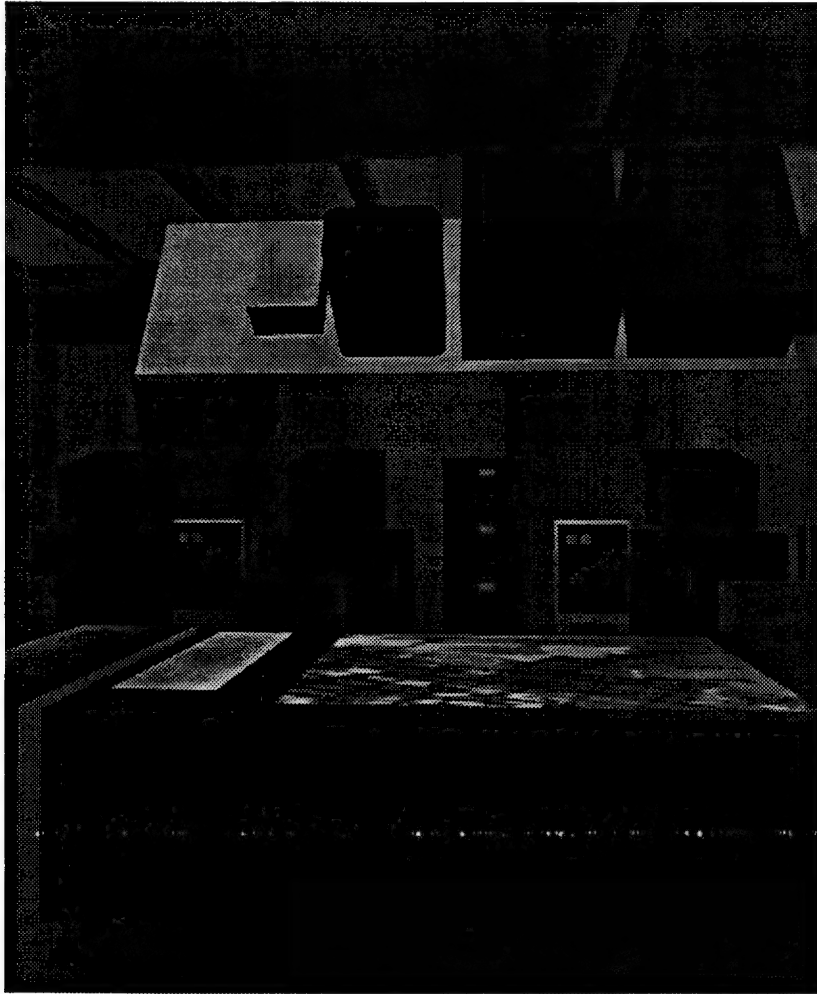


Figure 29: Chart table in CIC.

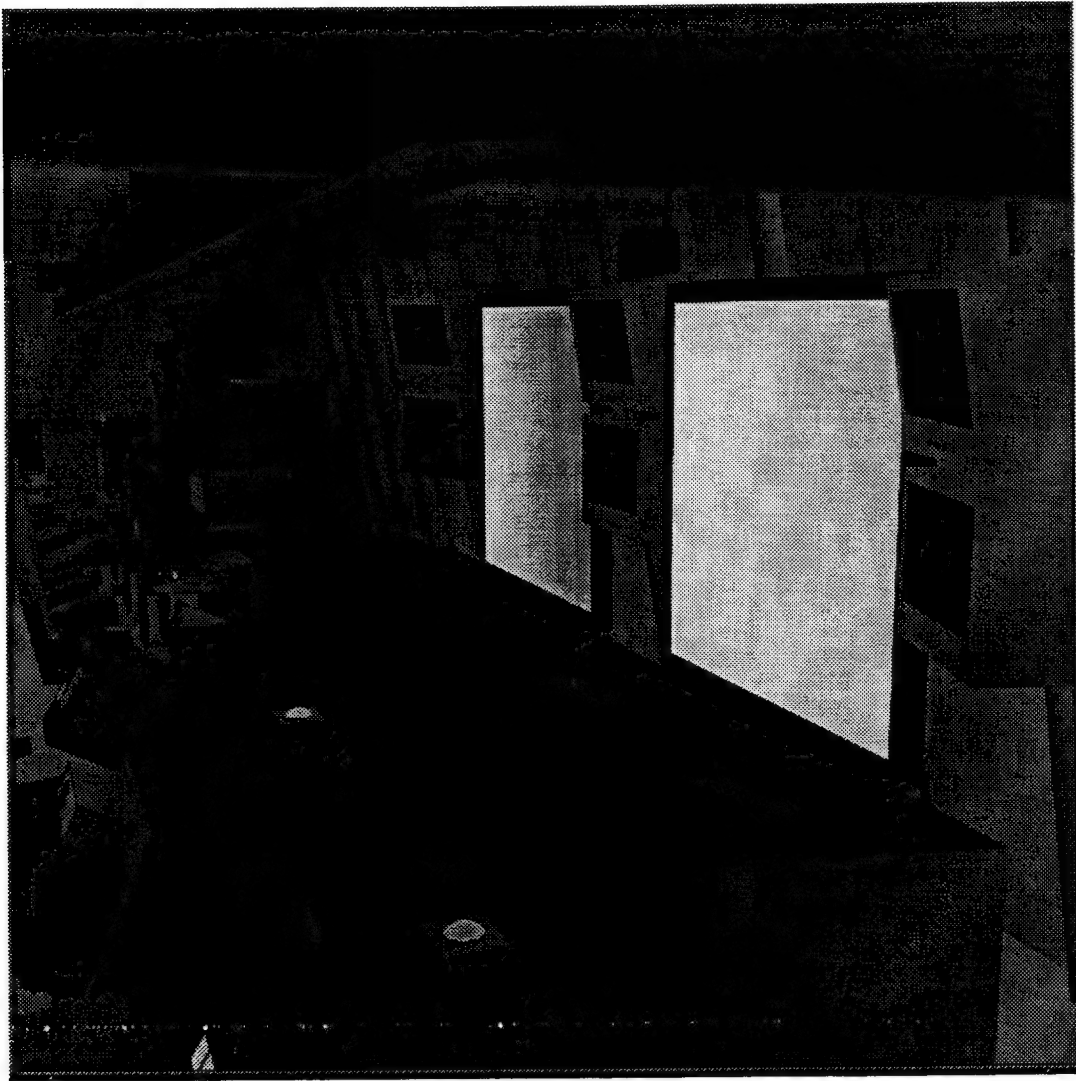


Figure 30: Command consoles in CIC.

VIII. CONCLUSION

A. RESULTS

The result of this work is a stand alone virtual environment capable of demonstrating the core requirements of any amphibious operation; namely, the ability to convey troops and equipment from mother ship to landing vehicle to operating area and back again. Although the amphibious simulator chose to accomplish this with the use of a surface landing vehicle (LCAC), the techniques and algorithms developed are general enough to enable air-borne operations as well.

Four main components went into the development of the amphibious simulator:

- EasyScene 3.0's high level API allowed quick development time, modularity, and real-time performance without loss of low level detail.
- Real-time collision detection was implemented on all mobile objects. This gave the amphibious simulator the capacity for realistic interactions between its entities. Humans can climb ladders and ramps, but won't go through walls. Ships react naturally when they run into each other. Just as important, collision detection was used to control the entity's mounting process.
- A generic mounting algorithm was implemented which makes possible dynamic changes in an entity's mounting hierarchy. This ability is the heart of the simulator and embodies the core amphibious requirements described above.
- Lastly, realistic models of the LPD-17 and LCAC were obtained and modified to support real-time rendering and well-deck operations. The models' high fidelity and detail were critical to a successful immersive experience.

B. FUTURE WORK

Although this is still a work in progress and is by no means complete, several possible uses for the amphibious simulator (and its successors) present themselves. Certainly a landing craft trainer comes to mind. This would allow craft masters/pilots to safely train in well-deck operations prior to going to sea. Also, vehicle loading and storage onboard amphibious ships is always a contentious issue. The use of an accurately scaled virtual environment in developing a loading plan would alleviate much of the problem. Another important example is the use of a simulation for planning the complex timing issues of large scale amphibious operations. Conflicts in air/water space as well as in time lines become readily apparent when the user is immersed in the scenario.

In order to be effective as a deployable system, however, much additional work is required. Some of the more important are listed here in order.

- Porting the simulation to NextGen and incorporate into NPSNET V. As a plug-in module, the amphibious simulator would have use of NPSNET V's built in network capability. This would allow the inclusion of a large number of entities, multiple users, as well as distance training (all participants would not need to be brought to the same location).
- Convert the models to VRML. The biggest obstacle to wide spread use of virtual environments as training aids is their dependence on specialized hardware and software. The revolutionary increases in PC performance and the equally impressive growth in VRML popularity will soon make it possible to bring these simulations to the desktop. Therefore, the quicker a simulation can be demonstrated on a PC, the more it will be viewed as a valid and cost effective solution to a problem.
- Construct improvements to the LPD-17 model. As mentioned in the previous chapter, work still needs to be done on the current model in order to facilitate implementation of a Potentially Visible Set and to allow the entire model to be used in real time.
- Realistic interfaces need to be acquired. The amphibious simulator will never reach its full training potential as a vehicle trainer until controls identical to their live vehicle

counterparts are integrated with the system. This is relatively low on the priority list since it only applies to a subset of the simulations possible uses. Walk through and planning applications are not effected by this since no mechanical skills are being exercised.

- Beaches and the near shore ocean floor need to be modeled. Beach gradients, mines, obstructions, and littoral currents are controlling factors during a landing operation. Their effects on vehicles and the planning process need to be implemented to increase realism.
- Lastly in terms of order of accomplishment but not in importance, is an empirical study of the benefits derived from this simulation.

APPENDIX

A. USER CONTROLS

The following table shows all the inputs needed to control the amphibious simulator.

The current ship is designated by clicking the left mouse button on the desired vehicle (LPD or LCAC).

Table 7: Amphibious simulator controls.

Key	Function
b	ballasts the LPD well-deck
g	opens and closes stern gate
l	manually lowers ramps on LCAC
r	resets the currently selected ship
s	displays simulator statistics
v	toggles from human to third person view
w	toggles models to and from wireframes
F1	sets sea state to 1 (default)
F2	sets sea state to 2
F3	sets sea state to 3
F4	sets sea state to 4
PAD8	accelerates human
PAD2	decelerates human
PAD6	turns human to the right
PAD4	turns human to the left
UPARROW	accelerates selected ship
DOWNARROW	decelerates selected ship
RIGHTARROW	turns selected ship to right

Table 7: Amphibious simulator controls.

Key	Function
LEFTARROW	turns selected ship to left

B. RUNNING THE SIMULATOR

- When first started the simulator's initial view is through the eyes of the human mounted onboard the LCAC.
- Use the UPARROW key to accelerate the LCAC toward the LPD's stern.
- Press the "g-KEY" to open the stern gate.
- Press the "b-KEY" to ballast the well-deck.
- Drive the LCAC straight into the open well-deck. The LCAC will automatically stop when it is grounded in the well deck.
- Press "b-KEY" again to ballast the ship back up.
- Use the PAD keys to navigate the human entity around the LCAC and walk off onto the LPD. Explore the vehicle storage areas and walk up to the flight deck.
- Once the human is off the LCAC it can be launched again, by ballasting the ship and backing the LCAC out of the well.

LIST OF REFERENCE

- [AIRE90] Airey, John M., Rohlf, John H., Brooks, Fredrick P. Jr., *Towards Image Realism with Interactive Update Rates in Complex Virtual Building Environments*, Computer Graphics, Vol. 24 No. 2, March 1990.
- [CRAI89] Craig, John J., *Introduction to Robotics*, Addison-Wesley Publishing Company, 1989.
- [CSIA96] Coryphaeus Software, Inc., *Easy Scene 3.0 API Reference*, P. Araki, W. Vogt, 1996.
- [CSIB96] Coryphaeus Software, Inc., *Easy Scene OpenSea Maritime Module 1.0 API Reference*, M. Buckiewicz, 1996.
- [GOSSW94] Gossweiler, Rich., Laferriere, Robert J., Keller, Michael L., Pausch, Randy., *An Introductory Tutorial for Developing Multiuser Virtual Environments*, Presence, Vol. 3, No. 4, 1994.
- [KING95] King, Tony E., McDowell, Perry L., *A Networked Virtual Environment For Shipboard Training*, Master's Thesis, Naval Postgraduate School, Monterey, CA, 1995.
- [LOCK95] Locke, John., *Applying Virtual Reality*, Unpublished paper, Naval Postgraduate School, Monterey, CA, 1995.
- [OBYR95] O'Byrne, James Edward, *Human Interaction Within a Virtual Environment For Shipboard Training*, Master's Thesis, Naval Postgraduate School, Monterey, CA, 1995.
- [PNA67] The Society of Naval Architects and Marine Engineers, *Principles of Naval Architecture*, J.P. Comstock, 1967.
- [PRAT93] Pratt, David R. , *A Characterization of Virtual World and Artificial Reality Systems*, Doctoral Dissertation, Naval Postgraduate School, Monterey, CA, 1993.
- [SGI94] Silicon Graphics, Inc., Document Number 007-1680-020, *IRIS Preformer Programming Guide*, J. Hartman, P. Creek, 1994.
- [STEW96] Stewart, Bryan C., *Mounting Human Entities to Control and Interact with Networked Ship Entities in a Virtual Environment*, Master's Thesis, Naval Postgraduate School, Monterey, CA, 1996.

BIBLIOGRAPHY

Fossen, Thor I., Guidance and Control of Ocean Vehicles. New York: John Wiley & Sons Inc., 1994.

Lentz, Fred., Shaffer, Alan., Pratt, David., Falby, John., Zyda, Michael. NPSNET: Naval Training Integration. Naval Postgraduate School, 1995.

Moshell, Michael., Cortes, Art., Clarke, Tom., Abel, Kimberly., Kilby, Mark., Lisle, Curtis., Maples, Daniel., Morie, Jacquelyn. Research in Virtual Environments and Simulation at the Institute for Simulation and Training of the University of Central Florida. *Presence*, Vol. 4, No. 2, 1995.

Pentland, Alex., Computational Complexity Versus Simulated Environments. Massachusetts Institute of Technology, Media Lab, 1990.

Stytz, Martin., Hobbs, Bruce., Kuntz, Andrea., Soltz, Brian., Wilson, Kirk., Portraying and Understanding Large-Scale Distributed Virtual Environments: Experience and Tentative Conclusions. *Presence*, Vol 4. No. 2, 1995.

Zyda, Michael., Osborne, William., Monahan, James., Pratt, Davis. NPSNET: Real-Time Vehicle Collisions, explosions and Terrain Modifications. *The Journal of Visualization and Computer Animation*, Vol. 4, No.1, 1993.

Zyda, Michael., Pratt, Davis., Falby, John., Lombardo, Chuck., Kelleher, Kristen. The Software Required for the Computer Generation of Virtual Environments. *Presence*, Vol. 2, No. 2, 1995.

INITIAL DISTRIBUTION LIST

1. Defense Technical Information Center.....2
 8725 John J. Kingman Rd., STE 0944
 Ft. Belvoir, VA 22060-6218

2. Dudley Knox Library.....2
 Naval Postgraduate School
 411 Dyer Rd.
 Monterey, CA 93943-5101

3. ECJ6-NP1
 HQ USEUCOM
 Unit 30400 Box 1000
 APO AE 09128

4. Chairman, Code CS2
 Computer Science Department
 Naval Postgraduate School
 Monterey, CA 93943-5000

5. Dr. Michael J. Zyda, Professor2
 Computer Science Department Code CS/ZK
 Naval Postgraduate School
 Monterey, CA 93943-5000

6. John S. Falby, Senior Lecturer2
 Computer Science Department Code CS/FJ
 Naval Postgraduate School
 Monterey, CA 93943-5000

7. LT Didier Le Goff1
 7 Dresser St #3A
 Newport, RI 02841-5000

8. Brian Hill1
 1725 Jefferson Davis Highway, Suite 1300
 Arlington, VA 22202

9. Marianne Nutting.....1
 Naval Sea Systems Command
 2531 Jefferson Davis Highway
 Arlington, VA 22242-5160